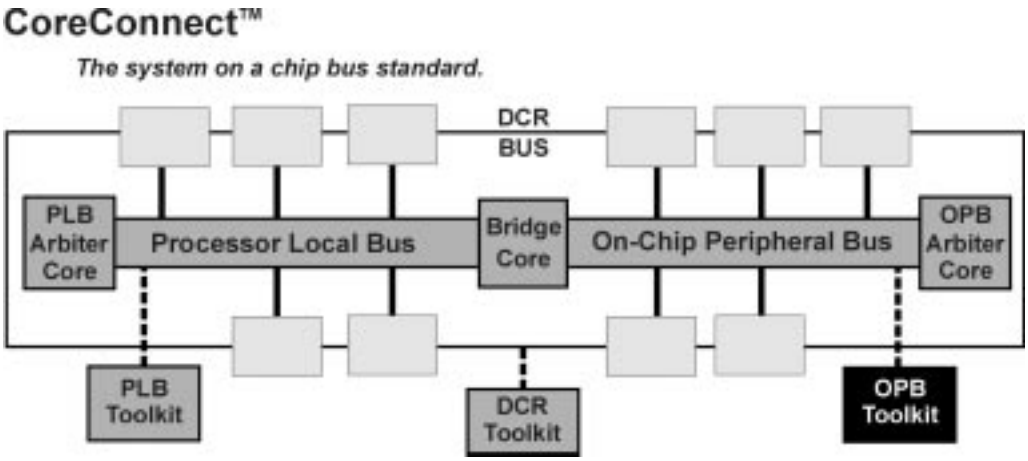
OPB Bus Functional Model Toolkit

User's Manual

Version 3.1



SA-14-2541-02

First Edition (May 2001)

This edition of On-chip Peripheral Bus Functional Model Toolkit User's Manual applies to the IBM OPB Bus Toolkit, until otherwise indicated in new versions or application notes.

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS MANUAL "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

IBM does not warrant that the products in this publication, whether individually or as one or more groups, will meet your requirements or that the publication or the accompanying product descriptions are error-free.

This publication could contain technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time.

It is possible that this publication may contain references to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only IBM's licensed program. You can use any functionally equivalent program instead.

No part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the written permission of IBM.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

Address comments about this publication to:

IBM Corporation
Department YM5A
P.O. Box 12195
Research Triangle Park, NC 27709

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1996, 2000. All rights reserved

4 3 2 1

Notice to U.S. Government Users – Documentation Related to Restricted Rights – Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

Patents and Trademarks

IBM may have patents or pending patent applications covering the subject matter in this publication. The furnishing of this publication does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, CT 06904, United States of America.

The following terms are trademarks of IBM Corporation:

IBM
CoreConnect
Other terms which are trademarks are the property of their respective owners.

Contents

Figures . . . . . **ix**
Tables . . . . . **xi**
About This Book . . . . . **xiii**
Chapter 1. OPB Toolkit Overview **1**
OPB Toolkit Features 2
OPB Bus Features 3
OPB Implementation 4
Chapter 2. OPB Toolkit Environment **6**
Bus Functional Compiler and Informational Files 6
Script Files 6
VHDL/Verilog Files 7
Chapter 3. OPB Toolkit Test Bench **8**
Instantiating Design Under Test 8
VHDL Signal Types 9
IEEE Packages 9
Chapter 4. OPB Bus Functional Compiler **10**
Compiler Simulator Configuration 10
Invoking the Bus Functional Compiler 10
Initializating the Bus Functional Models 10
Chapter 5. OPB Bus Models **12**
OPB Device Model 13
OPB Master Interface 14
OPB Slave Interface 15
OPB Data Interface 16
OPB DMA Interface 16
Model Operation 16
General Purpose Registers and ALU Instructions 17
Branch () Command 17
OPB Arbiter Model 17
OPB Arbiter Master Interface 18
Model Operation 18
OPB Monitor 19
OPB Monitor Interface 19
Chapter 6. OPB Bus Functional Language **21**
OPB Device Modes 21
Master Modes 21
Slave Modes 22
OPB Device Configuration Commands 22

Set_Device () Command 22
path = [string] 22
device_type = [string] 23
Alias Command 23
Set_alias () 23
Send () and Wait () Synchronization Commands 23
OPB Master Commands 24
Configure () Commands 24
Read () and Write () Bus Cycle Commands 26
Branch () Command 27
Move () Command 28
Compare () Command 28
Add () Command 29
Sub () Command 29
And () Command 29
Or () Command 30
Shift_left() Command 30
Shift_right() Command 30
Reg_Init () Command 31
Reg_Update () Command 31
OPB Bus Slave Commands 31
Mem_Init () Command 31
Configure() Commands 32
Response () Commands 33
Mem_Check () Command 34
Arbiter Model Configure () Commands 34
OPB Monitor Commands 34
Configure() commands 35
configure_report(), configure_read_report(), and configure_write_report() commands 35
Read() and Write() Monitor commands 36
Report() command 37
Chapter 7. OPB Bus Timing 39
Examples of BFL Command Files 39
Synchronized, Unlocked Multiple Master Memory Access 39
Locked Multiple Master Memory Access 40
Burst Write/Read with Lock and SeqAddr 41
Burst Write/Read with Lock and SeqAdd with Byte Enables 42
Write/Read with Transaction Checking and Reporting 43
Chapter 8. OPB Bus Compliance Checks 45
Terminology 45
OPB Signal Summary Table 45
OPB Master Interface Checks 47
OPB_MnGrant 47

OPB_timeout 47
Mn_busLock 47
Mn_select 47
Mn_RNW 48
Mn_hwXfer, Mn_fwXfer, Mn_dwXfer 48
Mn_seqAddr 48
Mn_UAbus (0:31) and Mn_ABus(0:31) 49
Mn_BE and Mn_BEXFER 49
Mn_DBus(0:63) 49
Mn_DBusEn 49
OPB_UAbus and OPB_ABus(0:31) 50
OPB_BE and OPB_BEXFER 50
OPB_DBus(0:63) 50
OPB_xferAck 50
OPB_hwAck 50
OPB_fwAck 51
OPB_dwAck 51
OPB_errAck 51
OPB_retry 51
OPB Slave Interface Checks 52
OPB_select 52
OPB_RNW 52
OPB_hwXfer, OPB_fwXfer and OPB_dwXfer 52
OPB_seqAddr 53
OPB_UABus(0:31) and OPB_ABus(0:31) 53
OPB_DBus(0:63) 53
Sln_xferAck 53
Sln_hwAck 53
Sln_fwAck 54
Sln_dwAck 54
Sln_errAck 54
Sln_retry 54
Sln_toutSup 55
Sln_DBus(0:63) 55
Sln_DBusEn 55
OPB Arbiter Interface Checks 55
OPB_busLock 55
OPB_toutSup 55
OPB DMA Interface Checks 56
DMA_SlnAck 56
Dynamic Bus Sizing (DBS) 56
64-bit Read Data Steering 59
Address/Control/Data Hold Checks 60
Index. 61

Figures

Figure 1. On-chip Peripheral Bus Interconnection 1
Figure 2. Physical Implementation of the OPB 4
Figure 3. OPB Toolkit Tesbench. 8
Figure 4. OPB Device Model 13
Figure 5. OPB Master Interface 14
Figure 6. OPB Slave Interface 15
Figure 7. OPB Data Interface 16
Figure 8. OPB DMA Interface 16
Figure 9. OPB Arbiter Master Interface 18
Figure 10. OPB Monitor Interface 20

Tables

Table 1. Summary of OPB Signals 45
Table 2. Unaligned Transfers Checked 48
Table 3. Unaligned Transfers Checked 52
Table 4. 64-bit Master Write Data Mirroring During Byte Enable Writes 56
Table 5. 64-bit Master Write Data Mirroring During Non Byte Enable Writes 58
Table 6. 64-bit Slave Read Steering 59

About This Book

This book begins with an overview followed by detailed information on On-chip Peripheral Bus Functional Model Toolkit environment, testbench, bus functional compiler, models and language used in simulation.

The On-chip Peripheral Bus Functional Model Toolkit features:

- Unit and system level simulation of logic designs compliant with OPB architectural specifications.
- VHDL and Verilog source models with simulator independence.
- Bus functional commands to generate and respond to variable transactions and delays.
- Bus protocol read/write checking and verification.
- Support for event and transaction synchronization.
- System performance modeling by using configurable models to represent typical device behavior.
- Faster simulation run-times than processor bus functional or full functional models.
- ‘What if’ simulation scenarios using different master or slave configuratiions.
- Verification and debugging of designs to assure bus compliance.

Who Should Use This Book

This book is for hardware, software, and application developers who need to understand Core+ASIC development and system-on-a-chip (SOC) designs. The audience should understand embedded system design, operating systems, and the principles of computer organization.

Since the OPB model toolkit was developed to comply with the OPB architectural specification, the toolkit users need to have a working level understanding of the architectural specification to be able to develop test cases and simulate using the bus model toolkit. The user should also be familiar with UNIX type operating systems, basic digital logic design and simulation, and the simulator which is used for the verification process.

Related Publications

The following publications contain related information:

Processor Local Bus Architecture Specifications

On-Chip Peripheral Bus Architecture Specifications

Device Control Register Bus Architecture Specifications

Processor Local Bus Toolkit User’s Manual

On-Chip Peripheral Bus Toolkit User’s Manual

Device Control Register Bus Toolkit User’s Manual

Processor Local Bus Arbiter Core User’s Manual

On-Chip Peripheral Bus Arbiter Core User’s Manual

PLB to OPB Bridge Core User’s Manual

OPB to PLB Bridge Core User’s Manual

How This Book is Organized

This book is organized as follows:

Chapter 1, “OPB Toolkit Overview”

Chapter 2, “OPB Toolkit Environment”

Chapter 3, “OPB Toolkit Test Bench”

Chapter 4, “OPB Bus Functional Compiler”

Chapter 5, “OPB Bus Models”

Chapter 6, “OPB Bus Functional Language”

Chapter 7, “OPB Bus Timing”

Chapter 8, “OPB Bus Compliance Checks”

To help readers find material in these chapters, the book contains:

- “Contents” on page v
- “Figures” on page ix
- “Tables” on page xi
- “Index” on page 61

Chapter 1. OPB Toolkit Overview

An on-chip peripheral bus (OPB) model toolkit provides unit and system level simulation and verification of ASICs and logic designs which comply with OPB architectural specifications. The toolkit enables the designer to accelerate the design cycle time by identifying and addressing possible problems at an earlier stage of the design cycle.

The on-chip peripheral bus (OPB) is designed for easy connection of on-chip peripheral devices. It provides a common design point for various on-chip peripherals. The OPB is a fully synchronous bus which functions independently at a separate level of bus hierarchy. It is not intended to connect directly to the processor core. The processor core can access the peripheral on this bus through the OPB bridge unit which is a separate core. The OPB model toolkit facilitates unit and subsystem level simulation of logic designs which are compliant with the OPB architecture. See the OPB architecture specifications for more information.

Figure 1 demonstrates how the on-chip peripheral bus is inter connected for the purpose of Core+ASIC development or system-on-a-chip design.

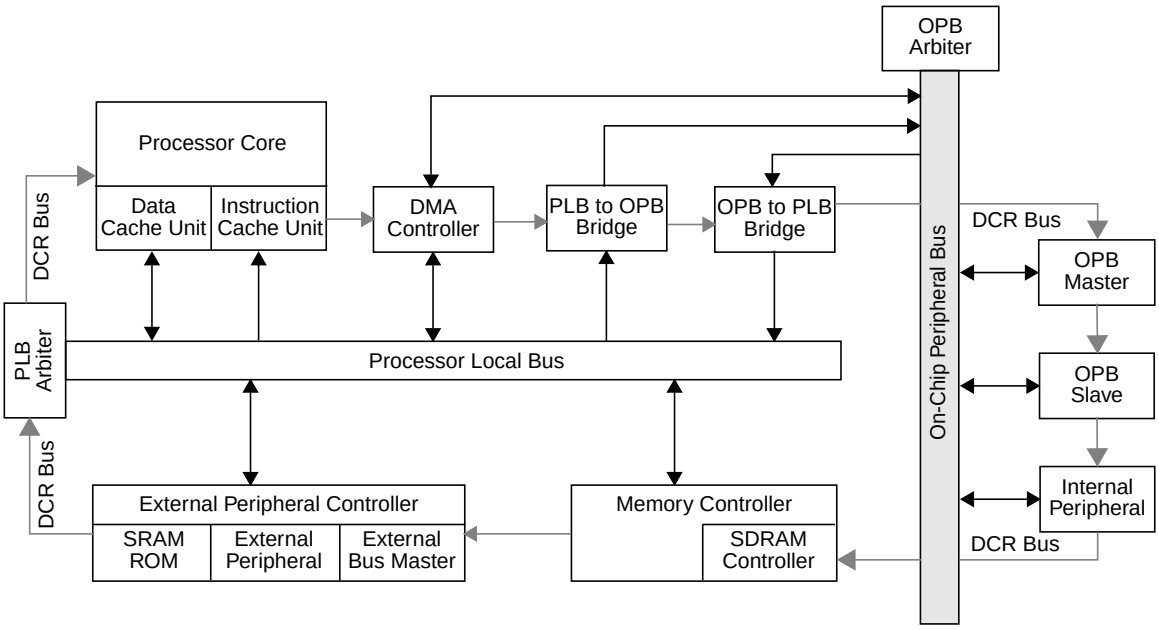


Figure 1. On-chip Peripheral Bus Interconnection

As shown in Figure 1, the on-chip bus structure provides a link between the processor core and other peripherals which consist of PLB and OPB master and slave devices.

The processor local bus (PLB) is the high performance bus used to access memory through the bus interface units. The two bus interface units shown above: external peripheral controller and memory controller are the PLB slaves. The processor core has two PLB master connections, one for instruction cache and one for data cache. Attached to the PLB is also the direct memory access (DMA) controller, which is a PLB master device used in data intensive applications to improve data transfer performance.

Lower performance peripherals (such as OPB master, slave, and other internal peripherals) are attached to the on-chip peripheral bus (OPB). A bridge is provided between the PLB and OPB to

enable data transfer by PLB masters to and from OPB slaves. In the above example we have two bridges, a PLB to OPB bridge which is a slave on the PLB and a master on the OPB and an OPB to PLB bridge which is a slave on the OPB and a master on the PLB. OPB peripherals may also comprise DMA peripherals.

The device control register (DCR) bus is used primarily for accessing status and control registers within the various PLB and OPB masters and slaves. It is meant to off-load the PLB from the lower performance status and control read and write transfers. The DCR bus architecture allows data transfers among OPB peripherals to occur independently from, and concurrent with, data transfers between the processor and memory, or among other PLB devices.

The OPB toolkit allows users to initiate OPB master cycles and provide slave responses through a bus functional language which is parameterized according to the architectural specification. Data checking and bus protocol monitoring also provide a way for users to automate the verification of OPB designs under development.

1.1 OPB Toolkit Features

Major features of the OPB toolkit consist of the following:

- Unit and subsystem level simulation of logic designs which comply with the OPB architectural specifications.
- VHDL and verilog source model solutions with simulator independence.
- Bus functional command definition to generate and respond to different transaction types with varying delays.
- Bus functional compiler which generates model initialization files from bus functional commands.
- Bus protocol checking through the use of general purpose bus monitors.
- Individual transaction checking through the monitor.
- Reporting mechanisms to report bus activity as seen by the bus monitor.
- Read and write data checking in masters and slaves.
- Model inter-communication bus for event and transaction synchronization.
- Enables peripheral developers to verify and debug their designs to assure bus compliance.
- Much faster simulation run-times than using PPC BFM or FFM to generate bus traffic.
- Allows ‘what if’ simulation scenarios using different master and slave configurations.
- Flexible, easy to use bus functional language (BFL) for quickly generating a variety of bus transactions.
- Provides hierarchical solution to verification.
- Master and Slaves can operate in a mixed byte enabled and non byte enabled environment.

1.2 OPB Bus Features

The OPB model toolkit enables the user to simulate the following OPB features.

- Address and data bus with automatic dynamic bus sizing in 16, 32, and 64 bit bus master requests
- A distributed multiplexer method of attachment where address and data buses are implemented in distributed AND-OR logic.
- Byte, halfword, and fullword duplication for byte, halfword, and fullword transfers.
- Single cycle transfer of data between OPB master and OPB slaves.
- Contiguous 8 bit byte enable support.
- Sequential address (burst) protocol support.
- Devices on the on-chip peripheral bus may be memory mapped, act as DMA peripherals, or support both transfer methods.
- 16-cycle fixed bus timeout provided by the OPB arbiter.
- OPB slave is capable of disabling the fixed timeout counter to suspend bus timeout error.
- Support for multiple OPB bus masters.
- Bus parking for reduced latency.
- OPB masters may lock the OPB bus arbitration.
- OPB slaves capable of requesting retry to break possible arbitration deadlock.
- OPB slaves capable of responding with an error acknowledge condition.
- Bus arbitration overlapped with last cycle of bus transfers.

1.3 OPB Implementation

Since the OPB supports multiple master devices, the address bus and data bus are implemented as a distributed multiplexer.

Control signals from OPB masters and slaves to and from the OPB arbiter and the peripherals will be similarly OR'ed together, and then sent to each device. Address and byte enable are multiplexed using the select, while data signals data bus enable signals driven by the OPB Devices. The OPB bus logic is implemented in the OPB toolkit test bench HDL which is described in more detail in "OPB Toolkit Test Bench" on page 8.

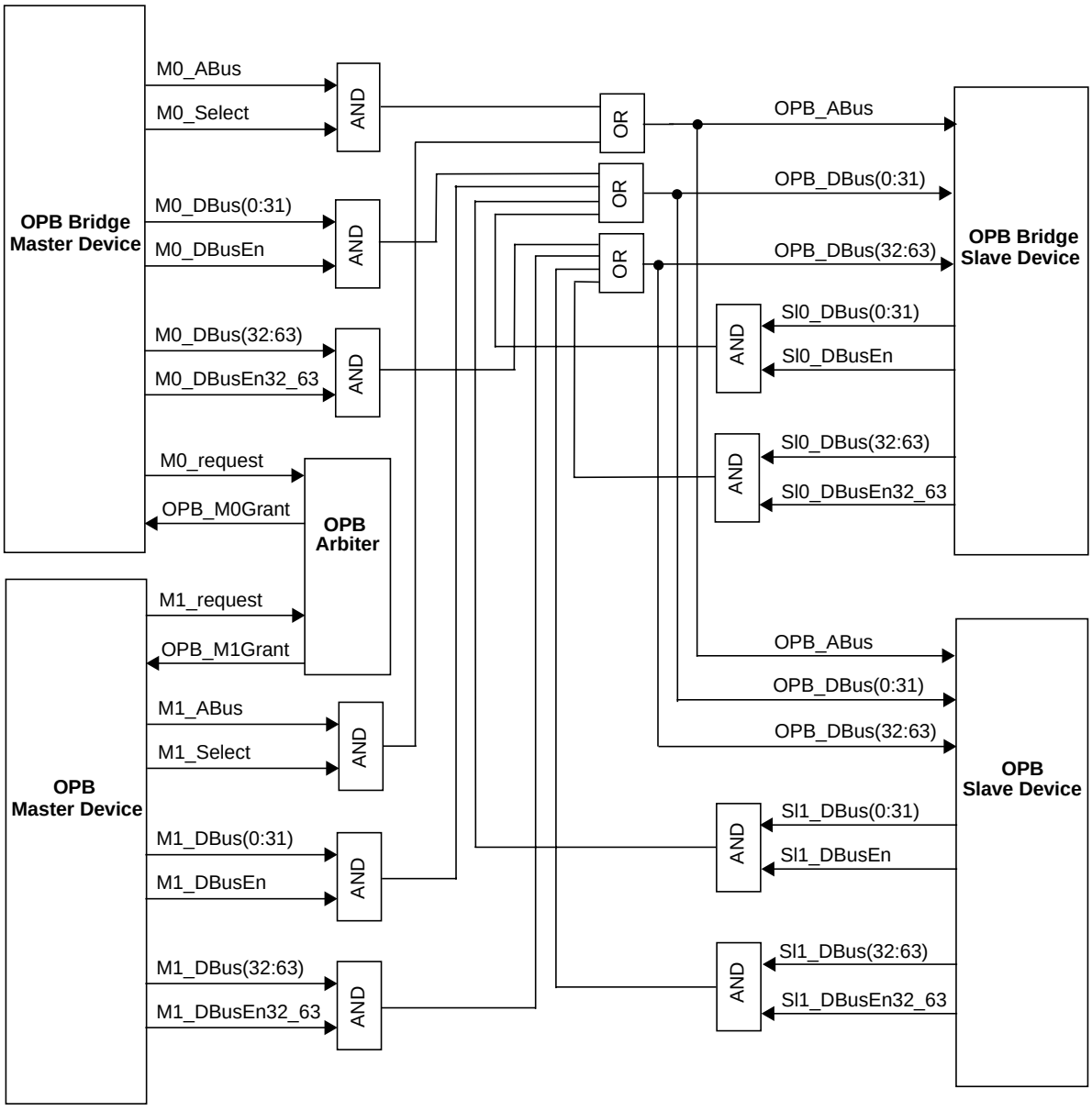


Figure 2. Physical Implementation of the OPB

Figure 2 shows a physical implementation of the OPB. Since the OPB supports multiple master devices, the address bus and data bus are implemented as a distributed multiplexer. This design will enable future peripherals to be added to the chip without changing the I/O on either the OPB arbiter or the other existing peripherals. By specifying the bus qualifiers as I/O for each peripheral (select for the ABus, DBusEn for the DBus), the bus can be implemented in a variety of ways, that is, as a distributed ring mux (shown below), using centralized AND/OR's or multiplexers, using transceiver modules and a "dotted" bus, etc. The optimal design for each implementation will vary, depending on the number of devices attached to the OPB and timing and routing constraints. \

Control signals from OPB masters and slaves to and from the OPB arbiter and the peripherals will be similarly OR'ed together, and then sent to each device. Bus arbitration signals such as Mn_request and OPB_MnGrant are directly connected between the OPB arbiter and each OPB master device.

Chapter 2. OPB Toolkit Environment

The OPB toolkit consists of a test bench, bus functional models, and a bus functional compiler. The test bench instantiates the bus models and design(s) under test. The toolkit is intended to provide users with an environment in which they may instantiate designs under test, create test cases, initialize models, and simulate with the ability to detect error conditions through both visual and automated bus protocol and data checking. Test cases are written in a bus functional language as described in “OPB Bus Functional Language” on page 21.

2.1 Bus Functional Compiler and Informational Files

The following files represent the bus functional compiler and a readme file.

- **BFC**

  This file is the Bus Functional Compiler. It is a Perl program that parses the test cases written in the bus functional language and generates command files which are used to control the bus functional models.

- **README**

  This file provides information about the toolkit release including functional support level and any known errata

- **UPDATE_LOG**

  This file provides information about functional changes and fixes between toolkit releases.

- **OPB_BFM_Workbook.pdf**

  This is the latest version of the OPB toolkit workbook.

2.2 Script Files

The following files are “EXAMPLE” script files for model compilation, invoking simulation, signal tracing, and a BFL test case.

- **analyze_VSS, analyze_MTI**

  These script files compile the VHDL version of the toolkit for VSS and MTI into the “work” library.

- **tcb**

  This Perl program invokes the bus functional compiler with a BFL file as a program argument. It then invokes the VSS simulator with the command file which was generated by the bus functional compiler.

- **sample.bfl, sample64.bfl, sample_probability.bfl, sample_branch.bfl, sample_slack.bfl, sample_auto.bfl, sample_seqaddr_MB.bfl**

  These are sample BFL test case files.

2.3 VHDL/Verilog Files

The following list of files comprise the VHDL OPB toolkit. The VHDL files should be compiled in the order specified:

- **opb_pkg.vhd/opb_pkg.inc**

  This HDL file is used by the OPB behavioral models to implement internal functions such as type conversion and other general procedures.

- **opb_dcl.vhd/opb_dcl.inc**

  This HDL file contains the component and constant declarations for the toolkit behavioral models.

- **opb_device.vhd/opb_device.v**

  This HDL file represents the I/O for OPB components. It instantiates a decode and bus unit which represent a master/slave OPB device. It handles the necessary conversions between standard logic and bit logic for all I/O signals

- **opb_bu_comp.vhd/opb_bu_comp.v**

  This HDL file represents the bus interface logic of the master/slave OPB device.It also contains the slave memory and slave decode logic for programmable bus responses.

- **opb_dc_comp.vhd/opb_dc_comp.v**

  This HDL file represents the decode and cycle issue logic of the OPB master.

- **opb_arb_comp.vhd/opb_arb_comp.v**

  This HDL file represents the arbitration logic for the OPB bus arbiter

- **opb_arb.vhd/opb_arb.v**

  This HDL file is a wrapper that instantiates the opb_arb_comp. It handles the necessary conversions between standard logic and bit logic for all I/O signals. Users should instantiate this entity in test fixtures which include the OPB model toolkit arbiter model.

- **opb_monitor_comp.vhd/opb_monitor_comp.v**

  This HDL file performs OPB cycle monitoring and protocol checking

- **opb_monitor.vhd/opb_monitor.v**

  This HDL file performs OPB cycle monitoring and protocol checking. It handles the necessary conversions between standard logic and bit logic for all I/O signals. Users should instantiate this entity in test fixtures which include the OPB model toolkit monitor model.

  IMPORTANT: All simulations should use the OPB Monitor to ensure OPB compliance.

- **opb_complex.vhd/opb_complex.v**

  This HDL file is the OPB test bench. Designs under test may be instantiated in the test bench and exercised with the OPB behavioral models.

Chapter 3. OPB Toolkit Test Bench

Figure 3 shows the default configuration of the OPB toolkit test bench environment. It contains three instantiations of OPB device model, OPB arbiter, and OPB bus monitor. It also contains the distributed multiplexers and logic to implement the OPB bus using AND-OR HDL statements. Designs under test are instantiated in the test bench HDL by the user.

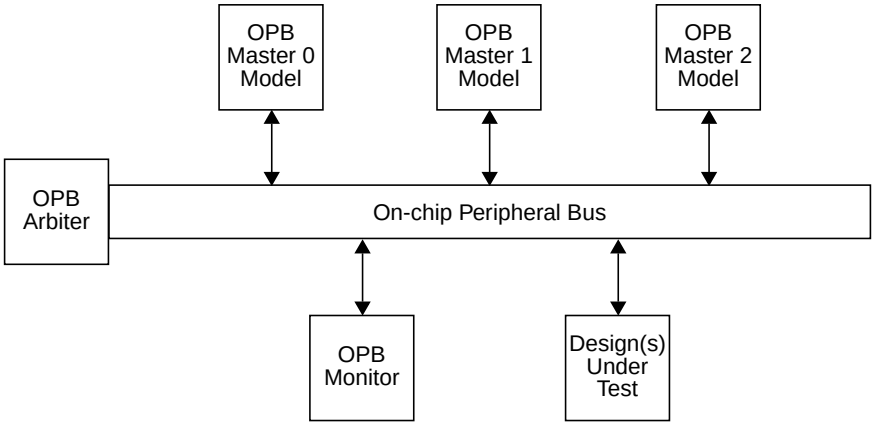


Figure 3. OPB Toolkit Tesbench

3.1 Instantiating Design Under Test

When instantiating a design under test, connect the OPB signals in the test bench to the device and ensure that the AND-OR OPB logic is updated to reflect the additional bus logic. This logic is located towards the bottom of the OPB test bench. In addition, if more than one design under test is added, update the max_opb_devices parameter in the OPB declarations file to accommodate more than four OPB devices. If the design under test is a slave, ensure that the OPB device address map has non-overlapping slave address space.

Note: When connecting smaller than 64 bit devices refer to the OPB 64-bit Architecture Specification for connection information.

The OPB device model instantiation generics/parameters have a default slave address mapping as follows, and may be changed in the generic mapping of the test bench:

Device 0	00000000-0000FFFF
Device 1	00010000-0001FFFF
Device 2	00020000-0002FFFF

The OPB device model has intercommunication signals which are called synch_in(0 to 31) and synch_out(0 to 31). The synch_in(0 to 31) is the logical OR of all the synch_out outputs of each instantiated OPB device model in the simulation environment. When instantiating OPB device models

which need to support the send/wait commands as described in “OPB Bus Functional Language” on page 21 ensure that the intercommunication logic within the test bench is updated to reflect the additional model instantiations.

3.2 VHDL Signal Types

The signal interface for the VHDL bus functional models and test bench are declared as IEEE std_logic and std_logic_vector signal types. If the bus functional models are integrated with a test bench environment which uses bit and bit_vector types, the wrappers may be eliminated so that type conversions do not have to be included in the model interfaces. When the wrappers are not used, please note that the I/O for each OPB toolkit model component contains primary inputs for configuration signals rather than the VHDL generic declarations which are in each wrapper.

3.3 IEEE Packages

The VHDL version of the OPB core RTL is translated from Verilog to VHDL with a source level language translator. The translated VHDL calls out some IEEE packages such as:

ieee.std_logic_1164.all;

ieee.std_logic_arith.all;

ieee.std_logic_unsigned.all;

Some VHDL compilers and analyzers may require the use of compiler directive switches to resolve overloaded relational operators.

Chapter 4. OPB Bus Functional Compiler

Test cases written in the OPB bus functional languages are parsed by the toolkit bus functional compiler. For VHDL simulators, it generates simulator interface commands to initialize the PLB models within the toolkit test bench. For Verilog simulators, the BFC generates a Verilog initialization file which contains Verilog statements to initialize the bus functional models. These command files are used to load the bus functional model command and data arrays after the test bench is loaded into the simulator.

The bus functional compiler is implemented in Perl, which is an interpreted language distributed under the GNU public license. It is available at no cost and runs on nearly all UNIX or UNIX-like operating systems. For more information about Perl, visit the Perl home page at http://www.perl.com/perl.

4.1 Compiler Simulator Configuration

The first time the BFC is executed, the user will be asked to answer a series of questions. These questions are used by the BFC in order to function correctly in the user's environment. The BFC will first ask the user to enter their default simulator. The user must type one of the listed simulators for the BFC to be configured. Remaining questions will contain a default or suggested answer noted within brackets. The user may either hit ENTER to accept the default, or type an alternative. Once configured, the BFC options may be changed at any time by hand editing the**.bfcrc** file or by deleting this file and reconfiguring the BFC.

4.2 Invoking the Bus Functional Compiler

The bus functional compiler operates on files with the extension ".bfl", and it generates a file with the same name but with a ".cmd" or ".do" or ".v" extension depending on the target simulator. If the ".bfl" extension is omitted, the ".cmd" extension is simply added to the end of the filename.

To invoke the bus functional compiler, type "BFC filename.bfl".

It is possible to invoke the BFC with multiple command files by specifying each file as an argument (input parameter) to the BFC. When multiple files are used in a single BFC call, the command file which is generated will be named using the first input file name.

To invoke the bus functional compiler, type "BFC filename1.bfl filename2.bfl...".

Note: If the message "perl: not found" or other system error is encountered when invoking the BFC, ensure that the path for the Perl executable is correct on the first line of the BFC source program as required by the Perl interpreter specification. To locate the Perl executable, try using the UNIX command "which perl."

4.3 Initializating the Bus Functional Models

The command files which are generated by the bus functional compiler should be executed at simulation time 0. For the VHDL toolkit this is accomplished with an "include" or "do" command when invoking the simulator using the simulator command interface. For the Verilog toolkit the BFC

generates a Verilog initialization command file which should be included when the simulation model is compiled.

Chapter 5. OPB Bus Models

The toolkit contains models for OPB devices, an OPB arbiter, and an OPB monitor.

The OPB device model contains both master and slave function. It is controlled through a bus functional command interface which is defined in “OPB Bus Functional Language” on page 21. The master issues requests for the bus and generates cycles. The slave decodes the OPB cycles and responds as necessary.

The OPB arbiter model receives all the master request signals and asserts master grant signals based on a configurable arbitration algorithm. This algorithm may be configured as one of the supported arbitration algorithms. The behavioral algorithms include random, round robin, and priority implementations.

The OPB monitor model receives most of the signals in the OPB test bench environment. It performs bus protocol checking for arbitration and data transfers. Error detection is reported to the user through direct register access and message generation.

Figure 4 illustrates the architecture of the OPB toolkit device model.

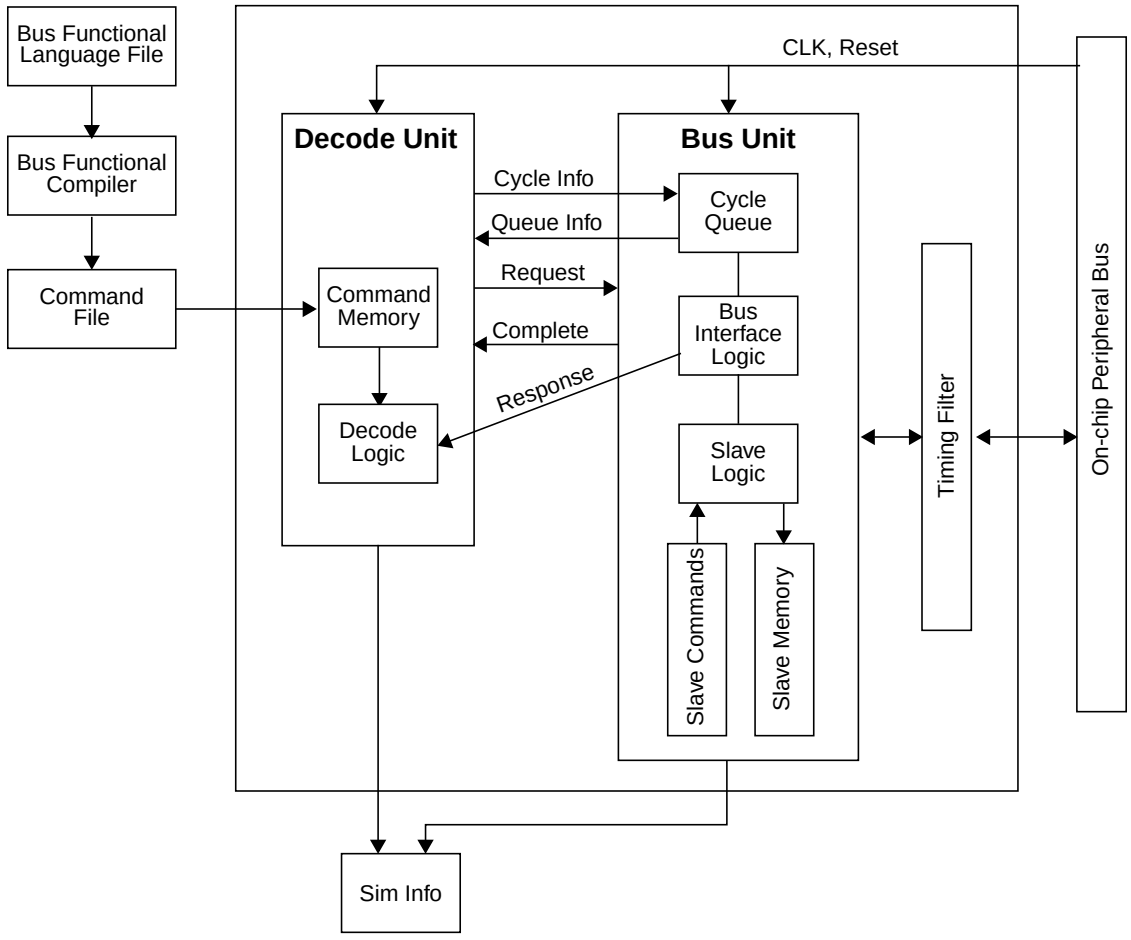


Figure 4. OPB Device Model

5.1 OPB Device Model

An OPB device model may act as a master or a slave, or as both. The master contains logic to automatically request the bus when it has commands to execute. Once the arbiter issues the grant signal, it performs an operation based on the bus functional command which the user initialized. The slave responds to cycles based on an address decode or DMA operation, and it maintains an internal memory which can be initialized through the bus functional language. This memory may be dynamically checked during simulation, or when all bus transactions have completed.

The following signals are used to connect OPB device models on the OPB.

5.1.1 OPB Master Interface

Figure 5 shows all master interface signals. These signals are used to connect masters on the OPB bus. OPB master device can also act as OPB slave device for other OPB bus master request. See OPB architecture specifications for detailed functional description.

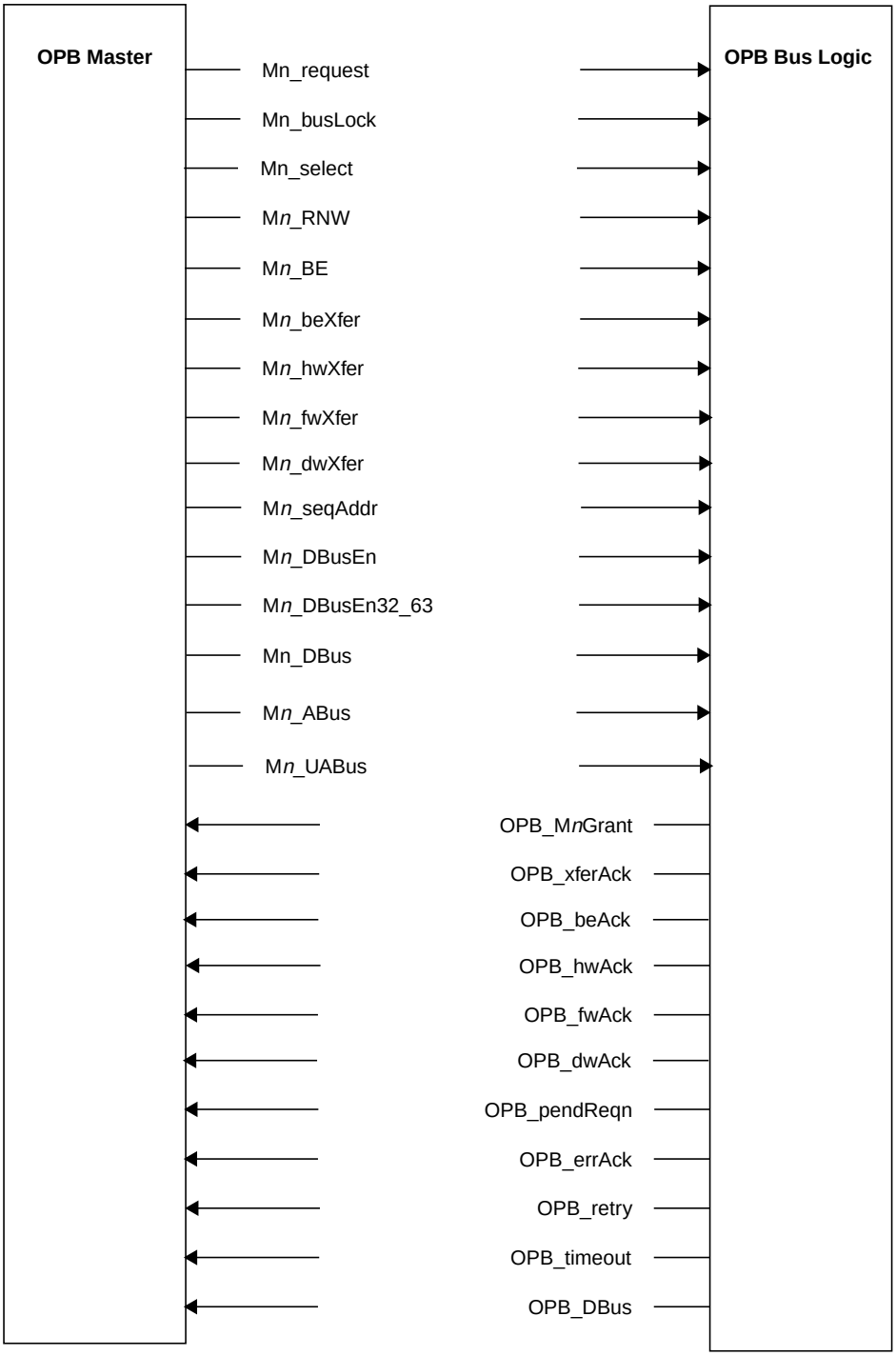


Figure 5. OPB Master Interface

5.1.2 OPB Slave Interface

Figure 6 shows all OPB slave interface signals. These signals are used to connect slave devices on the OPB bus. This diagram also describes a fullword device with all 32 bits of OPB_DBus and Sln_DBus connected. Slave devices may also be of byte (8bit) or halfword (16 bit) widths. See OPB architecture specifications for detailed functional description

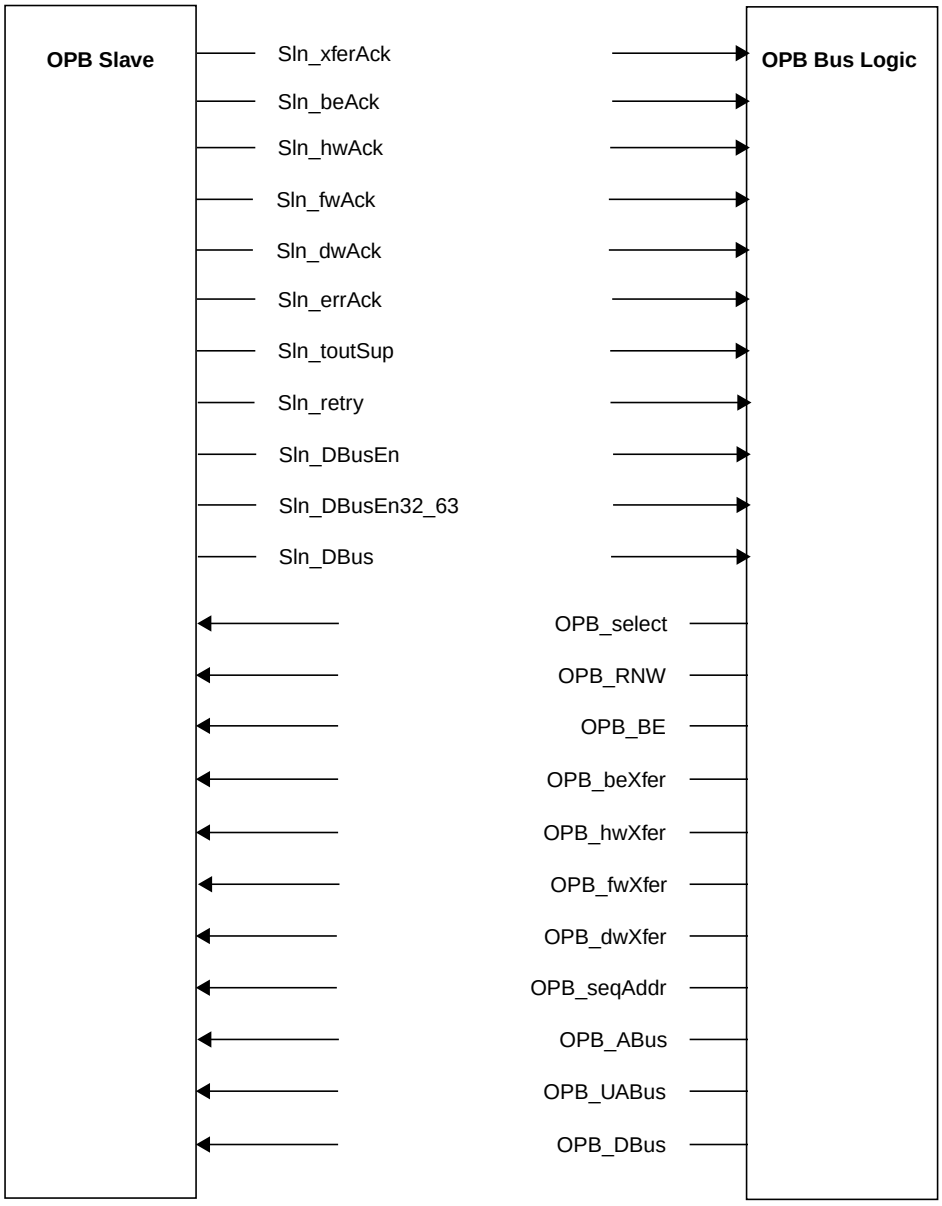


Figure 6. OPB Slave Interface

5.1.3 OPB Data Interface

Figure 7 shows all OPB data interface signals. See OPB architecture specifications for detailed functional description.

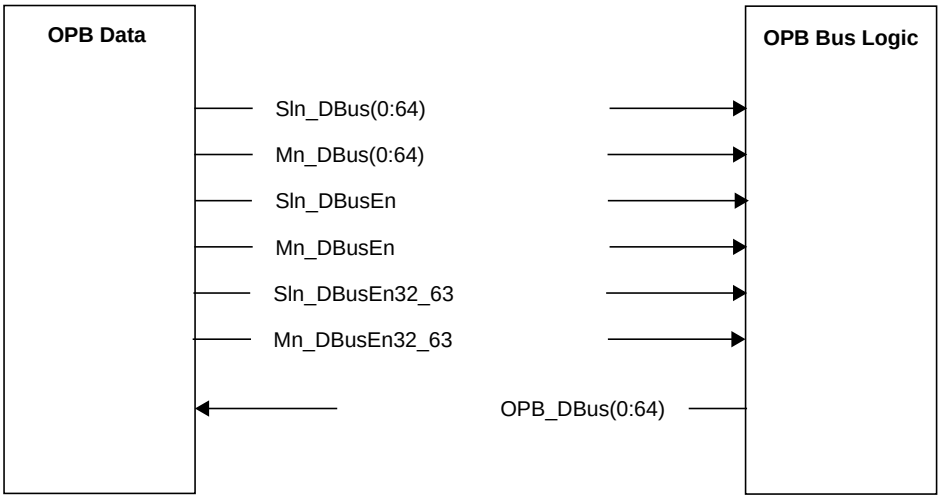


Figure 7. OPB Data Interface

5.1.4 OPB DMA Interface

Figure 7 shows all OPB DMA interface signals. See OPB architecture specifications for detailed functional description.

Figure 8. OPB DMA Interface

5.1.5 Model Operation

The OPB device model executes bus commands from internal command arrays which are loaded from the command file (or core) generated by the bus functional compiler. When invoking the simulator, include the command file so that the command arrays are initialized when the simulation model is loaded. This is typically accomplished by using an include file parameter or core execution with most simulators. The master and slave response commands are executed sequentially during simulation. The decode unit can issue bus requests every OPB clock if there is a free entry in the cycle queue. A request delay parameter may suspend the issuance of a command to the bus unit until a delay counter has expired. All memory initialization and configuration parameters are also done at

simulation time 0. Intermediate data checking during simulation can be done by using memory check commands which are executed when an intercommunication signal is received by the slave. Bus master read data is checked on a cycle basis.

When an OPB_timeout is received by a master, it generates a warning message and proceeds to the next instruction or bus cycle if dynamic bus sizing is occurring. When an OPB_retry is received, the master reissues the cycle that was retried.

5.1.6 General Purpose Registers and ALU Instructions

The OPB master model contains 32 general purpose registers (R0-31) which are 32 bits wide. It also contains a 32-bit synchronization register (SR) and a 32-bit condition register (CR). These registers can be used by the programmer to implement algorithms or conditions which affect the way in which the OPB master executes commands. In addition, the following commands are available for arithmetic operations:

- Add()
- Sub()

The condition register is updated during ‘compare’ instructions. It provides a mechanism for testing and branching. The bits of the Condition Register are grouped into eight 4-bit fields, named CR Field 0 (CR0)..., CR Field 7 (CR7). Instructions are provided to perform logical operations on individual CR bits and to test individual CR bits. The bits of the CR “fields” are interpreted as follows:

- bit 0: Negative (LT) The result is negative.
- bit 1: Positive (GT) The result is positive.
- bit 2: Zero (EQ) The result is zero.
- bit 3: undefined

The synchronization register latches the synch_in input, and it may be cleared with a MOVE instruction.

Note: When an interrupt signal is connected to the SYNCH_IN signal of the OPB Master model, the synchronization register may be used with a compare instruction to alter program execution.

5.1.7 Branch () Command

The OPB master model contains a branch instruction so that users can implement loops and forward jumps. The branch instruction uses the condition register to determine whether a jump is taken.

5.2 OPB Arbiter Model

The OPB arbiter model simulates arbitration algorithms similar to the way an actual OPB arbiter logic design may control the bus. Therefore, the arbiter model which is provided with the toolkit can be considered a “full functional” model since it does not decode and execute instructions from a bus functional command interface. A user may choose to substitute other arbitration algorithms by configuring the test bench with an alternate arbiter which more closely models the target system environment.

5.2.1 OPB Arbiter Master Interface

Figure 9 shows all OPB arbiter master interface input/output signals. These signals are used to connect OPB bus arbiter to the OPB bus. See OPB architecture specifications for detailed functional description

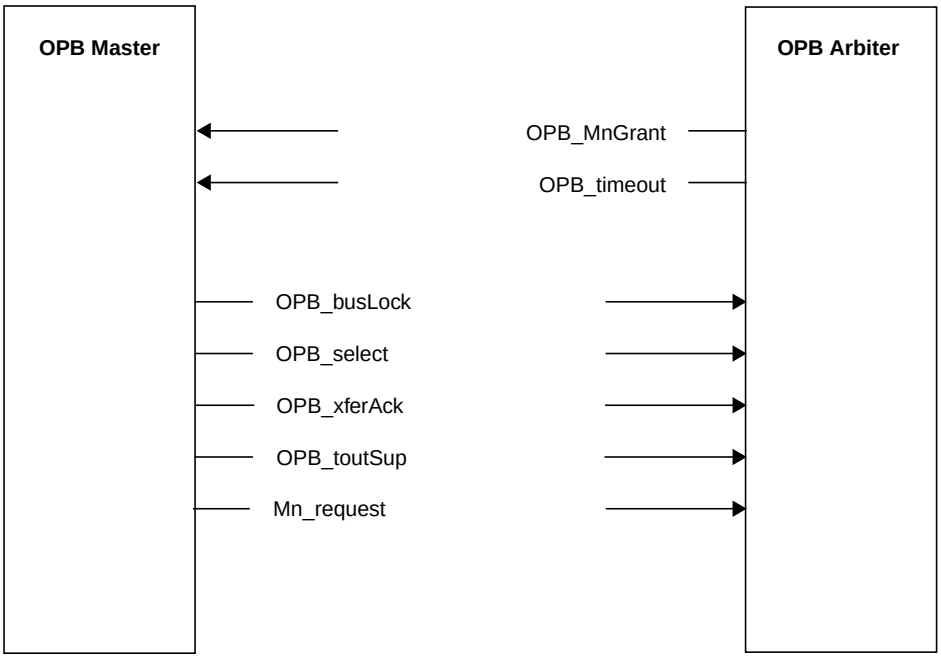


Figure 9. OPB Arbiter Master Interface

5.2.2 Model Operation

The OPB behavioral arbiter model implements three general purpose arbitration algorithms which can be selected from a bus functional language configuration command. The three supported algorithms are: priority, round_robin, and random. In addition, the behavioral arbiter supports bus parking with configurable park mode and park master attributes.

The three different arbitration algorithms are described in the following sections:

- Priority Mode

  When the arbiter is configured in priority mode, it always grants the bus to the requesting master with the highest index in the Mn_request signal. The master connected to M0_request has the lowest priority, and the master connected to Mn_request has the highest priority during an arbitration cycle. For example, when multiple masters are requesting the bus in the same arbitration cycle, the master with the highest integer index in the Mn_request signal will always be granted the bus.

- Round Robin Mode

When the arbiter is configured in round robin mode, it grants the bus to the requesting master with the next sequential index in the Mn_request signal with respect to which master was last granted the bus. For example, if master 0 is granted the bus, and master 0, 2, and 3 are requesting the bus in the next arbitration cycle, master 2 will be granted the bus.

- Random Mode

  When the arbiter is configured in random mode, it may grant the bus to any device which is requesting the bus.

5.3 OPB Monitor

The OPB monitor is a model which monitors the OPB by continuously sampling the OPB signals. It checks for violations of architectural specifications and reports warnings and errors to the user. Refer to section 8.3 for explanation of checks performed by the monitor.

5.3.1 OPB Monitor Interface

Figure 6 shows all OPB slave interface signals. See OPB architecture specifications for detailed functional description.

X - indicates the number of master devices attached to the OPB.

Y - indicates the DMA channel number that the slave device is attached to.

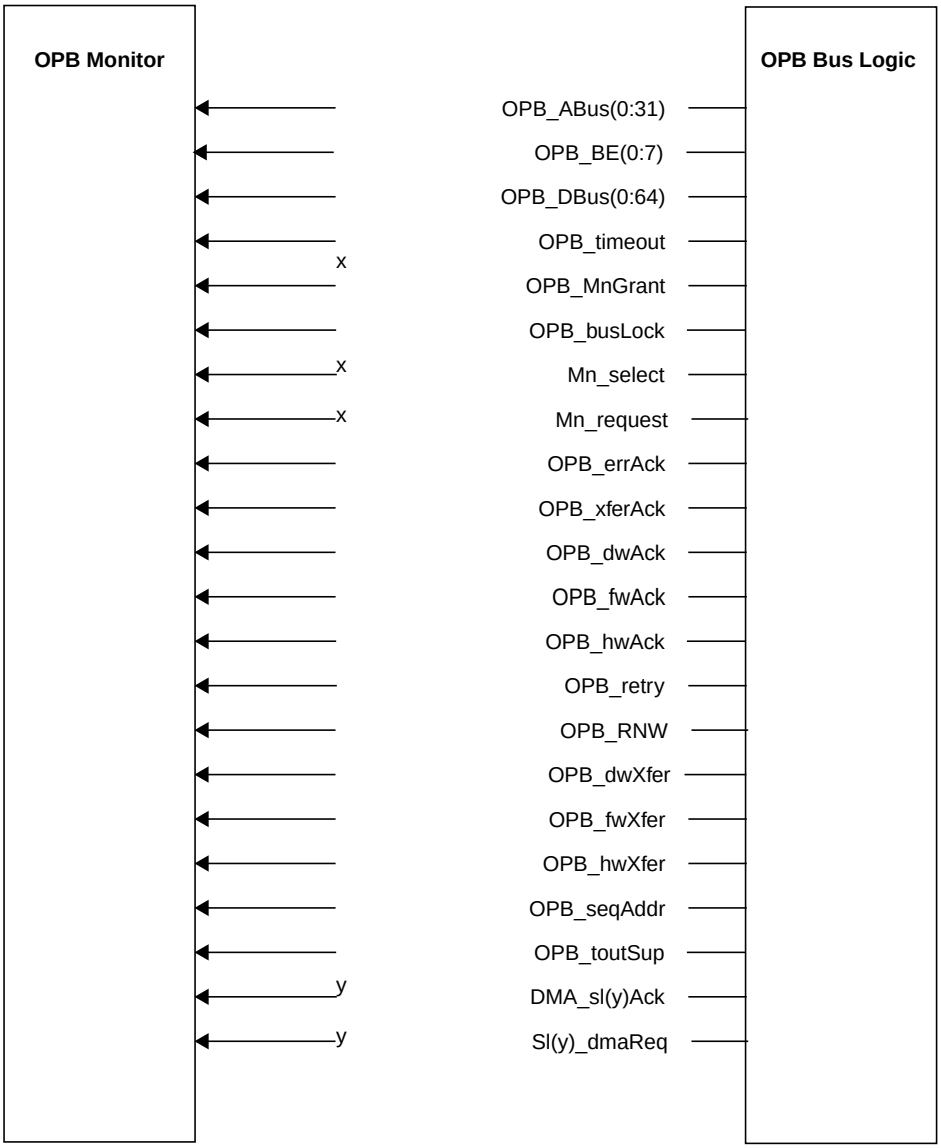


Figure 10. OPB Monitor Interface

Chapter 6. OPB Bus Functional Language

The bus functional models in the OPB model toolkit can be controlled through command files which contain information on how to initiate and respond to bus cycles. The command files also contain model configuration information. The general form of a bus functional language command is specified as follows: *command ([parameters]).*

Parameters define attributes for bus cycles and configurations. A bus command may have more than one parameter and have the following format: *PARAMETER = value*. A parameter value may be either a scalar or enumerated type (i.e. a string).

Note 1: BFL comments are delineated using the “--” or “//” string. All characters after the comment delimiter are ignored by the BFL parser. An end of line character terminates a comment.

Note 2: The bus functional language is case insensitive.

Note 3: A * denotes an optional parameter, otherwise the parameter is required.

Note 4: There is no restriction on the order in which parameters may be specified within a command.

Note 5: If multiple parameters are specified for the same command, they must be separated by commas.

The following sections describe the bus functional language for each of the bus models.

6.1 OPB Device Modes

This section discusses the possible modes that are available for OPB Master and Slave Devices. OPB Masters may be configured to act in either Command mode or Auto mode, while OPB Slave devices may be configured to act in one of three modes: Command, Configure, or Auto Mode. An explanation for the functionality of each of these modes is given below.

6.1.1 Master Modes

- **Command Mode** - In this mode the user provides read/write commands for a master for each transaction that they want to execute. In addition to the individual commands, the user may optionally specify parameter values within a configure statement that will apply to all read/write commands.
- **Auto Mode**(random) - In this mode the user will set the master_auto_mode to true in the configure statement.

Note: The default for this parameter is false and thus does not need to be specified if the user does not want to use random mode.

Note: To avoid read data comparison errors from user defined slave macros, errack_read_data_check_disable should be set to 1in the configure statement of the respective master.

The master_auto_mode will cause the toolkit to randomly generate varied transactions for the master. Along with the master_auto_mode parameter, the user may also specify other parameter values within the configure statement that will apply to the generated transactions. These optional parameters are useful for eliminating undesired transaction types. A complete listing of optional values is given in the OPB Master Commands section, along with their default values.

6.1.2 Slave Modes

- **Command Mode** - In this mode the user provides a response command for each transaction that requires a response. In addition to the individual commands, the user may optionally specify parameter values within a configure statement that will apply to all read/write commands.

Note: When in command mode, configuring of ack_size and ack_type will be ignored.

- **Configure Mode** - In this mode, no response statements are necessary. The master or slave is configured to respond in the same manner for each transaction. The configure statement is used to allow the user to achieve a specific type of response by providing parameter values that will apply to each response. If no parameterized configure statement is provided the slave will default to responding as a byte device with default values of zero for all optional parameters.
- **Auto Mode**(random) - In this mode the user will set the slave_auto_mode to true in the configure statement.

  Note: The default for this parameter is false and thus does not need to specified if the user does not want to use random mode.

The slave_auto_mode will cause the toolkit to randomly generate varied transactions for the slave. Along with the slave_auto_mode parameter, the user may also specify other parameter values within the configure statement that will apply to the generated responses. These optional parameters are useful for eliminating undesired response types. A complete listing of optional values is given in the OPB Slave Commands section, along with their default values.

6.2 OPB Device Configuration Commands

This section discusses set_device (), path = [string], and device_type = [string] configuration commands.

6.2.1 Set_Device () Command

The set_device command selects an OPB device model to initialize. Within a BFL file (test case), there should only be one set_device () command used per model initialization. If more than one set_device () command is used for a single model instantiation, all or part of the previously specified command section will not be used.

6.2.2 path = [string]

The string specifies the path of the model within the test bench hierarchy. The commands following the set_device command are used to initialize the specified device. Additional set_device commands are used to specify the initialization of other devices.

6.2.3 device_type = [string]

The string specifies the type of model being initialized. The valid strings for this toolkit are OPB_device or OPB_arbiter. Any mismatch in the device_type parameter and the model being initialized with cause an initialization error.

6.3 Alias Command

6.3.1 Set_alias ()

The set_alias is an optional command which sets up an alias to be used for string substitution in the bus functional command parameters. The aliases must be set up before they are used. Multiple aliases may be set up. It is possible to have alias files separate from BFL command files by invoking the BFC with multiple BFL files. The syntax for the set_alias command is set_alias(target_name=string_name)

- **target_name = [string]**

  This string specifies the name of the alias. Whenever the string is found in the command list, the BFC will substitute the alias value with the alias name.

- **string_value= [string]**

When an alias is established, the BFC automatically substitutes the string_value for every occurrence of the target_name.

Example: set_alias(TARGET_REG = 01020304)

In this example, the BFC automatically replaces every occurrence of the string “TARGET_REG” with string “01020304”.

6.4 Send () and Wait () Synchronization Commands

The send and wait commands are high level model synchronization commands which allow the user to control when commands are executed. They do not directly cause OPB activity, but signal between multiple OPB device model instantiations in order to coordinate bus activity.

The send command causes a vectored intercommunication signal to be asserted at a corresponding level parameter. The wait command causes the bus master to suspend instruction decode until an intercommunication signal is received. The send signal is asserted for one clock per send instruction. The send and wait instructions are executed sequentially along with the read/write commands. The send instruction waits for all previously issued bus cycles to complete on the bus before sending its intercommunication signal.

- **level = [integer: range 0 to 31]**

  The level parameter allows for one or more send signals to one or more OPB device models.

  Note: It is possible for a user to use the same intercommunication level with multiple masters, however the assertion of the same level in the same clock by multiple masters will not be able to be distinguished on the intercommunication send vector.

  Note: Multiple levels may be used in the same clock by listing more than one level within the send() or wait() command and separating the levels by commas. Example: send(level=0,level=2)

6.5 OPB Master Commands

This section discusses the configure, read/write bus cycle, configure, ALU, and restart commands.

Note: A cycle is defined as the time it takes for a bus transaction/command to complete.

6.5.1 Configure () Commands

The configure command allows the user to configure different OPB model attributes. It is important to note that the configure commands are only executed at time 0 in simulation.

- *** master_addr_LO_x = [hexadecimal: 4 or 8 bytes] default: 00000000_00000000**
- *** master_addr_HI_x = [hexadecimal: 4 or 8 bytes] default: 00000000_0000FFFF**

  These parameters allow a user to override the default generic address decode parameters for the OPB device masters. The user may provide up to two non-contiguous address ranges for the same master by varying the ‘x’ from 0 to 1. If more non-contiguous master address ranges are necessary, the user may instantiate additional OPB device models in the test bench.

- **master_auto_mode = [boolean] default: false**

  This parameter specifies whether the OPB master behavioral should automatically generate bus cycles. This automatic generation is referred to as “random mode” or “automatic mode”. The valid types are true or false. When the master model is configured for automatic mode, there is no need to initialize read/write commands since they are ignored when the bus cycle generation is enabled.

- **errack_read_data_check_disable = [integer: 0 to 1]**

  This parameter is used to disable read checks when a toolkit master is in master_auto_mode. See Note in Master Modes section.

- **m_byte_enable[boolean] default:false**

  This parameter is used with the master_auto_mode and determines whether byte enables are generated. A setting of true causes the master to generate byte enabled transactions.

- *** auto_max_cycle = [integer: 0 to 4095] default: 48**

  This parameter specifies the number of bus cycles that will be generated by an OPB master behavioral under random conditions.

- *** seed = [integer] default: none**

  This parameter specifies the seed to be used for the random features of the model. The user may vary this value to vary the slave responses given by the model.

- *** master_auto_data = [boolean: true or false] default:false**

  This parameter specifies whether the OPB master behavioral should automatically generate data. This parameter works with both command and random mode. When the master model is configured for automatic data mode, there is no need to initialize internal memory data since it is generated and checked automatically from within the BFM.

- *** data_seed = [integer] default:none**

This parameter specifies the seed to be used for the auto data feature of the model. The master model forms a seed using this data_seed and the byte address. The user may vary the initial seed to vary the random number sequences which are generated, which in turn will produce varying master data.

- *** lock_probability = [integer: range 1 to 4095] default: 15**

  This parameter is used under random conditions to specify the probability that lock will be asserted. It follows that the lower the value given to this parameter, the higher the occurrence of a buslock. This means that a value of ‘1’ will result in a buslock for all randomly generated requests and a value of ‘2’ will result in a buslock for approximately half of all randomly generated requests.

- *** unlock_mode = [enumerated type: clk, cycle] default: cycle**

  This parameter specifies the mode in which lock is deasserted. This may be specified as clock mode or cycle mode. Clock mode specifies the deassertion of lock as the number of clocks from when lock was previously asserted. Cycle mode specifies the deassertion of lock during the last cycle of the number of cycles specified with the unlock parameter.

- *** unlock_min = [integer: range 1 to 4095] default: 1**
- *** unlock_max = [integer: range 1 to 4095] default: 1**

  This parameter is used with random mode to specify the lowest and highest integer values that can be chosen for “unlock” by the random number generator. The “unlock” parameter specifies the number of clocks to deassert buslock after it was previously asserted, or the number of cycles to elapse before it is deasserted.

- *** seq_probability = [integer: range 1 to 4095] default: 10**

  This parameter is used under random conditions to specify the probability that seq will be asserted. It follows that the lower the value given to this parameter, the higher the occurrence of a seqaddr.

- *** unseqaddr_mode = [enumerated type: clk, cycle] default: cycle**

  This parameter specifies the mode in which seqaddr is deasserted. This may be specified as clk mode or cycle mode. Clk mode specifies the deassertion of seqaddr as the number of clocks from when seqaddr was previously asserted. Cycle mode specifies the deassertion of seqaddr during the last cycle of the number of cycles specified with the unseqaddr parameter.

- *** unseqaddr_min = [integer: range 1 to 4095] default: 1**
- *** unseqaddr_max = [integer: range 1 to 4095] default: 1**

  This parameter is used with random mode to specify the lowest and highest integer values that can be chosen for unseqaddr by the random number generator. The “unseqaddr” parameter specifies the number of clocks to deassert Mn_seqAddr after it was previously asserted.

- *** req_delay_min = [integer] default: 0**
- *** req_delay_max = [integer] default: 10**

This parameter is used with random mode to specify the lowest and highest integer values that can be chosen for the “req_delay” by the random number generator. The “req_delay” parameter specifies how many clocks for the master to wait before it asserts its request signal for the corresponding read or write cycle.

6.5.2 Read () and Write () Bus Cycle Commands

The read/write bus cycle commands initiate read or write cycles on the OPB by causing the bus master to request the bus by asserting its Mn_request signal. The Mn_select signal is asserted when the OPB arbiter grants the bus to the bus master initiating the read or write command. These commands are executed sequentially, therefore the completion of the cycles on the OPB influence the simulation time at which the commands are decoded. The following is a list of possible read/write parameters:

- **addr = [hexadecimal: 4 or 8 bytes] default: none**

  This parameter specifies the 32/64 bit address.

- **be = [8 bit] default: none**

  This parameter specifies the Mn_BE (byte-enable) signals of the OPB. The valid byte_enable combinations are listed in the OPB architecture specifications in section 5.5.1.2. req_size is not required when using a byte-enable capable master. The req_size will be calculated by the BFC based on the bit pattern entered for the vector.

- **req_size = [integer: 1, 2, 4, or 8] default: none**

  This parameter specifies the full word/half word request signals of the OPB. They generate the encoding for fwXfer/hwXfer as defined in the OPB architecture specifications. The valid req_size integer values are 1, 2, or 4 bytes.

- **data = [hexadecimal: 1,2,4, or 8 bytes] default: none**

  This parameter represents the write data to be generated or the read data to be checked on read cycles. The data value must be specified in hex format.

- *** req_delay = [integer: 0 to 4095] default: 0**

  This parameter specifies how many clocks for the master to wait before it asserts its request signal for the corresponding read or write cycle. The request delay is counted from the clock after grant is asserted in response to a master request. However, a request delay in the first master command will be counted from the assertion of the bus_request signal, which should be two clocks after opb_reset is deasserted.

- *** lock = [integer: range 0 to 1] default: 0**

  This parameter specifies a lock condition of the OPB bus. If the parameter is not specified, lock is not asserted. If the cycle is terminated before the lock signal is asserted, the lock parameter is ignored. When using the lock parameter, an unlock parameter is required in the same instruction. The lock signal is automatically deasserted if a retry is received.

- *** unlock = [integer: range 1 to 4095] default: 1**

  This parameter specifies the number of clocks to deassert buslock after it was previously asserted, or the number of cycles to elapse before it is deasserted. The mode for the deassertion of lock is specified with the previous unlock_mode parameter.

- *** unlock_mode = [enumerated type: clk, cycle] default: cycle**

This parameter specifies the mode in which buslock is deasserted. This may be specified as clock mode or cycle mode. Clock mode specifies the deassertion of lock as the number of clocks from when lock was previously asserted. Cycle mode specifies the deassertion of lock during the last cycle of the number of cycles specified with the unlock parameter.

- *** seqAddr = [integer: 0] default: none**

  This parameter allows for the assertion of seqaddr in the same clock that select is asserted. An error message is generated for any seqaddr value other than zero. If the parameter is not specified, seqAddr is not asserted. The architecture specifies that lock must be high when seqaddr is high, therefore the lock signal is forced high with the assertion of the seqaddr. It is the responsibility of the user to ensure that cycle addresses in the BFL are specified as sequential, however, there is a special force_seqaddr parameter available that will override non-sequential addresses with sequential addresses(See entry for “force_seqaddr”) when seqaddr is high. When using the seqaddr parameter, an unseqaddr parameter is required in the same instruction.

- *** unseqAddr = [integer: range 1 to 4095] default: 1**

  This parameter specifies the number of clocks to deassert seqaddr after it was previously asserted. If the lock signal is deasserted before the number of clocks specified expire, the seqaddr signal is automatically deasserted.

- *** unseqAddr_Mode = [enumerated type: clk, cycle] default: cycle**

  This parameter specifies the mode in which seqaddr is deasserted. This may be specified as clk mode or cycle mode. Clk mode specifies the deassertion of seqaddr as the number of clocks from when seqaddr was previously asserted. Cycle mode specifies the deassertion of seqaddr during the last cycle of the number of cycles specified with the unseqaddr parameter.

- *** force_seqaddr = [integer: 0 or 1] default: 0**

  This parameter is used to override non-sequential addresses with sequential addresses when seqaddr is high. This parameter was specifically designed to be used with the CoreConnect Test Generator to allow testing of seqaddr.

- *** deselect= [integer: range 1 to 4095] default: none**

  This parameter specifies the number of clocks to wait before deasserting the select signal. This allows the user to terminate cycles before a response is received from a slave. If the parameter is not specified, select will be deasserted normally, at the completion of a transfer.

- *** ack_size = [enumerated type: 1, 2, 4, or 8] default: none**

  This parameter specifies the expected full word/half word acknowledge signals of the OPB. The master compares the expect with the actual bus acknowledge signals and generates an error message upon a mismatch. The valid ack_size integer values are 1, 2, 4 or 8 bytes.

6.5.3 Branch () Command

The branch command allows the user to re-direct the command decode location within the OPB master decode unit internal command array. Looping and jumping can be accomplished using this instruction. Branch targets are established by specifying label attributes with a OPB master command.

Note: Branching between devices is not supported.

For example:

T1: read(req_delay=3,addr=00001000,req_size=4,be=1111,data=AABBCCDD) --establish a target called T1

Note that the ":" between the label and master command delineates the two BFL constructs. Each branch instruction executes in one OPB clock cycle.

branch(SRC,OP,Label)

- **SRC**

  This parameter specifies the condition register to be used to determine whether the branch is taken. A condition register is used in this comparison.

- **OP**

  This parameter specifies the comparison operator to compare the condition register. The valid conditions are:

  LT: less than

  GT: greater than

  EQ: equal

  NL: not less than

  NG: not greater than

  NE: not equal

- **Label**

  This parameter specifies the BFL destination of the branch if the condition register matches the operator parameter. The label must defined in the test case using the label command attribute.

  Example: branch(CR3,LT,Label1)

6.5.4 Move () Command

The move command allows the user to move a 32 bit internal register or memory value to another internal register. Each move instruction executes in one OPB clock cycle.

move(DST=SRC) or move(DST,SRC)

- **SRC = [CR, SR, R0 - R31]**

  The SRC parameter specifies the internal register or 32-bit internal memory value to be used as the source of the move instruction.

- **DST = [CR, SR, R0 - R31]**

  The DST parameter specifies the internal register which will updated with the SRC parameter value. The DST may be CR, SR, or R0-31.

  Example: move(CR=R0) or move(CR,R0)

6.5.5 Compare () Command

The compare command allows the user to compare two values and store the results in the condition register. Each compare instruction executes in one OPB clock cycle.

compare(SRC1,SRC2,DST)

- **SRC1, SRC2 = [SR, R0 - R31]**

  These parameters specify the internal register values to be used as the source of the compare instruction. Each compare instruction requires two SRC parameters.

- **DST**

  This parameter specifies the condition register destination of the compare instruction. Valid values are CR0-7.

  Example: compare(R0,R1,CR1)

6.5.6 Add () Command

The add command updates an internal master register with the result of an addition between an internal register and an immediate value. Each add command executes in one OPB clock cycle.

add(DST=Value) or add(DST,Value)

- **DST = [R0 - R31, SR, or CR]**

  This parameter specifies the 32-bit internal destination register to be updated. The DST must be R0-31, SR, or CR.

- **Value = [hexadecimal: up to 4 bytes]**

  This parameter specifies the immediate hexadecimal value to be added to the destination register. Example: add(R0,01)

6.5.7 Sub () Command

The sub command updates an internal master register with the result of a subtraction between an internal register and an immediate value. Each sub command executes in one OPB clock cycle.

sub(DST=Value) or sub(DST,Value)

- **DST = [R0 - R31, SR, or CR]**

  This parameter specifies the 32 bit internal destination register to be updated.

- **Value = [hexadecimal: up to 4 bytes]**

  This parameter specifies the immediate hexadecimal value to be subtracted from the destination register.

  Example: sub(R0,01)

6.5.8 And () Command

The AND command updates an internal master register with the result of a bit-wise AND operation between an internal register and an immediate value. Each AND command executes in one OPB clock cycle.

and(DST=Value) or and(DST,Value)

- **DST [R0 - R31, SR, or CR]**

This parameter specifies the 32 bit internal destination register to be updated.

- **Value = [hexadecimal: up to 4 bytes]**

  This parameter specifies the immediate hexadecimal value to be AND'ed with the destination register.

  Example: and(R0,FFFFFFF0)

6.5.9 Or () Command

The OR command updates an internal master register with the result of a bit-wise OR operation between an internal register and an immediate value. Each AND command executes in one OPB clock cycle.

or(DST=Value) or or(DST,Value)

- **DST = [R0 - R31, SR, or CR]**

  This parameter specifies the 32 bit internal destination register to be updated.

- **Value = [hexadecimal: up to 4 bytes]**

  This parameter specifies the immediate hexadecimal value to be OR'ed with the destination register.

  Example: or(R0,01)

6.5.10 Shift_left() Command

The Shift_left command updates an internal master register with the result of a bit-wise SHIFT LEFT operation between an internal register and an immediate value. Each SHIFT LEFT command executes in one OPB clock cycle.

shift_left(DST=Value) or shift_left(DST,Value)

- **DST = [R0 - R31, SR, or CR]**

  These two parameters specify the 32 bit internal destination register to be updated.

- **Value = [integer value]**

  This parameter specifies the immediate integer value to indicate the number of bit positions to shift the specified register.

  Example: shift_left(R0,1)

6.5.11 Shift_right() Command

The Shift_right command updates an internal master register with the result of a bit-wise SHIFT RIGHT operation between an internal register and an immediate value. Each SHIFT RIGHT command executes in one OPB clock cycle.

shift_right(DST=Value) or shift_right(DST,Value)

- **DST = [R0 - R31, SR, or CR]**

  This parameter specifies the 32 bit internal destination register to be updated.

- **Value = [integer value]**

  This parameter specifies the immediate integer value to indicate the number of bit positions to shift the specified register.

  Example: shift_right(R0,1)

6.5.12 Reg_Init () Command

The reg_init command initializes an internal master register and is only executed at simulation time 0.

reg_init(DST=Value) or reg_init(DST,Value)

- **DST = [R0 - R31, SR, or CR]**

  This parameter specifies the master 32 bit register to initialize.

- **Value = [hexadecimal: 4 byte]**

  This parameter specifies the 32 bit value that will be assigned to the master register at time 0.

  Example: reg_init(R0=01020304)

6.5.13 Reg_Update () Command

The reg_update command updates an internal master register during the decode and execution of master bus commands. Each reg_update command executes in one OPB clock cycle.

- **DST = [R0 - R31]**

  This parameter specifies the 32 bit destination register of the internal register to be updated.

- **Value = [hexadecimal: 4 byte]**

  This parameter specifies the 32 bit value that will be assigned to the internal register at execution time.

  Example: reg_update(R0=05060708)

  Example: compare(R0,R1,CR1)

  Example: or(R0,01)

6.6 OPB Bus Slave Commands

This section discusses the mem_init, configure, response, and mem_check commands.

6.6.1 Mem_Init () Command

The mem_init command initializes slave memory and it is done at time 0 in simulation

- **addr = [hexadecimal: 4 or 8 bytes]**

  This parameter specifies the 32 or 64 bit address.

- **data = [hexadecimal: 4 bytes]**

  This parameter specifies the slave memory data to initialize in the corresponding OPB device.

6.6.2 Configure() Commands

- *** slave_addr_LO_x = [32 bit] default: 00000000_00000000**
- *** slave_addr_HI_x = [32 bit] default: 00000000_0000FFFF**

  These parameters allow a user to override the default generic address decode parameters for the OPB device slaves. The user may provide up to two non-contiguous address ranges for the same slave by varying the ‘x’ from 0 to 1. If more non-contiguous slaves address ranges are necessary, the user may instantiate additional OPB device models in the test bench. It is important that there are no overlapping memory address areas between multiple slaves, since both slaves would attempt to respond to a single cycle. If both address parameters are equal, they are not used as a valid address decode.

- ***s_byte_enable [boolean] default:false**

  This parameter specifies whether the slave will respond as a byte-enable capable slave.

- ***ack_size = [integer: 1,2,4, or 8] default:1**

  This parameter specifies the default slave bus size, which may be a 1, 2 or 4 byte device.

- *** slave_auto_mode = [boolean: true or false] default: false**

  This parameter specifies whether the OPB master behavioral should automatically generate bus cycles. The valid types are true or false.

- *** seed = [integer] default:1**

  This parameter specifies the seed to be used for the random features of the model. The user may vary this value to vary the slave responses given by the model.

- *** slave_auto_data = [boolean] default:false**

  This parameter specifies whether the OPB slave behavioral should automatically generate data. When the slave model is configured for automatic data mode, there is no need to initialize internal memory data since it is generated and checked automatically from within the BFM.

- *** data_seed = [integer] default:none**

  This parameter specifies the seed to be used for the auto data feature of the model. The slave model forms a seed using this data_seed and the byte address. The user may vary the initial seed to vary the random number sequences which are generated, which in turn will produce varying slave data.

- *** normal_probability = [integer: 0 to 100] default: 25**
- **errack_probability = [integer: 0 to 100] default: 25**
- **retry_probability = [integer: 0 to 100] default: 25**
- **timeout_probability=[integer: 0 to 100] default:25**

  These parameters specify the probabilities for each of the acknowledge types when the auto mode feature of the model is being used. The slave model will produce acknowledge types based on the values given to each of the above parameters. This allows the user to eliminate the occurrence of an entire acknowledge type(s), as well as determine how often a particular acknowledge type(s) will occur. The values given to each must add up to 100. The example below shows a case in which the slave will produce timeouts and normals equally, but will not produce erracks or retries.

  Example 1:

The example below will produce an even mix of normal and timeout responses.

set_device (path=/opb_complex/opb_device2,device_type=opb_device) -- initialize slave 2

configure (normal_probability=50, timeout_probability=50, retry_probability=0, errack_probability=0, slave_auto_mode=true, seed=421, slave_auto_data=true, data_seed=3071)

Example 2:

The example below will produce an even mix of all types, which is the default if no values are specified.

set_device (path=/opb_complex/opb_device2,device_type=opb_device) -- initialize slave 2

configure (normal_probability=25, timeout_probability=25, retry_probability=25, errack_probability=25, slave_auto_mode=true, seed=421, slave_auto_data=true, data_seed=3071)

- **slave_ack_size_min=[integer] default:0**
- **slave_ack_size_max=[integer] default:2**

  These parameters specify the minimum and maximum slave response sizes when the slave auto mode is being used. The range is 0 to 3, with 0 being a byte response, 1 being a halfword response, 2 being a fullword response, and 3 being a doubleword response. The ack_size will vary for each transaction and will always fall within the specified range.

  Example:

  set_device (path=/opb_complex/opb_device2,device_type=opb_device) -- initialize slave 2

  configure(slave_auto_mode=true, slave_ack_size_min=0, slave_ack_size_max=2, delay=0, slave_auto_data=true, data_seed=2375, seed=23)

- **slave_max_wait_cnt_min=[integer: 0 to 4095] default:0**
- **slave_max_wait_cnt_max=[integer: 0 to 4095] default:14**

  These parameters specify the minimum and maximum time to wait before a transfer is terminated when the slave auto mode is being used. The value of the slave_max_wait_cnt will vary for each transaction and will always fall within the specified min and max range.

  Example:

  set_device (path=/opb_complex/opb_device0, device_type=opb_device) -- Initialize slave 0

  configure(slave_max_wait_cnt_min=0, slave_max_wait_cnt_max=5, slave_auto_mode=true, seed=326)

6.6.3 Response () Commands

The response command specifies attributes for a slave cycle response when an address decode is true.

- *** ack_Size = [integer: 1, 2, 4, or 8] default: 1**

  This parameter specifies the slave full word/half word response to be 1, 2, or 4 bytes as defined in the OPB architecture specifications. If the Ack_size parameter is omitted, the configured slave_bus_size is used for the slave bus response. This may have been initialized with the configure command. The valid ack_size values are 1, 2, or 4 bytes.

- **ack_Type = [enumerated type: normal,retry,error,timeout] default: normal**

  This parameter specifies the type of transfer termination which is defined as follows:

  normal: this response type generates a normal slave cycle termination.

  retry: this response type generates a retry cycle termination.

  error: this response type asserts the errAck signal with the xferAck signal.

  timeout: this response type causes no generation of any acknowledge signal.

- *** delay = [integer: 0 to 4095] default: 0**

  This parameter specifies the number of clocks to wait before terminating the transfer. Note that this counter automatically accounts for the assertion of the suppress signal.

Note: A delay of 16 will cause a OPB timeout.

6.6.4 Mem_Check () Command

The mem_check command automatically compares slave memory with the specified comparison data. If there is a data comparison error, an error message is generated and the corresponding error detection bit is set in the OPB device.

- **level = [integer: 0 to 31]**

  This parameter specifies which synchronization signal to wait for before checking the slave memory data.

- **addr = [hexadecimal: 4 or 8 bytes]**

  This parameter specifies the 32 or 64 bit address.

- **data = [hexadecimal: 4 or 8 bytes]**

  This parameter specifies the slave memory data to check in the corresponding OPB device.

6.6.5 Arbiter Model Configure () Commands

The configure command allows the user to configure different OPB model attributes. It is important to note that the configure commands are only executed at time 0 in simulation.

- *** arbiter_mode = [enumerated type: priority, round, random] default: round**

  This parameter specifies the arbitration mode for the OPB behavioral arbiter.

- *** arbiter_park_enable = [boolean: true or false] default: false**

  This parameter specifies whether the OPB behavioral arbiter should park the OPB.

- *** arbiter_park_master = [integer:0 to max_opb_devices-1] default:0**

  This parameter specifies which OPB device to park on when parking is enabled.

6.7 OPB Monitor Commands

This section discusses the read, write, configure, report, configure_write_report, and configure_read_report commands.

6.7.1 Configure() commands

The configure command allows the user to configure different OPB monitor attributes. These configure commands are only executed at time 0 in simulation.

- ***unlock_mode = [enumerated type: clk, cycle] default: cycle**

  This parameter specifies the mode in which the deassertion of lock is checked by the monitor. This may be specified as clk mode or cycle mode. Clock mode specifies to check the deassertion of lock based upon the number of clocks (from when lock was previously asserted) that have as specified by the unlock parameter. Cycle mode specifies to check the deassertion of lock based upon the number of transactions (or cycles), from when lock was previously asserted, that have been specified by the unlock parameter.

- ***unseqaddr_mode = [enumerated type: clk, cycle] default: cycle**

  This parameter specifies the mode in which the deassertion of seqaddr is checked by the monitor. This may be specified as clk mode or cycle mode. Clock mode specifies to check the deassertion of seqaddr based upon the of number of clocks from when unseqaddr was previously asserted (specified by the unseqaddr parameter). Cycle mode specifies to check the deassertion of unseqaddr based upon the number of transactions (or cycles), from when unseqaddr was previously asserted, that have been specified by the unseqaddr parameter.

6.7.2 configure_report(), configure_read_report(), and configure_write_report() commands

These commands allow the user to configure, or filter, the types of transactions that are reported during simulation. The configure_report command applies to all transactions that are reported through the monitor. The configure_read_report command applies to all read transactions that are reported through the monitor, while the configure_write_report command applies to all write transactions. Without any commands defining ranges, the monitor will be able to report all transactions.

- ***addr_min = [32 bit] default: none**
- ***addr_max = [32 bit] default: none**

  These parameters are used to specify the range of addresses that are to be reported by the monitor. Transactions with addresses outside of this range will be discarded.

- ***data_min = [64 bit] default: none**
- ***data_max = [64 bit] default: none**

  These parameters are used to specify the data range that are to be reported by the monitor. Transactions with data outside of this range will be discarded.

- ***acksize = [enumerated type: 1, 2, 4 and 8] default: none**

  This parameter is used to specify that transactions with the specified acknowledge size are to be reported. Multiple acksize parameters may be used to include a combination of acknowledge sizes. Transactions with acksize values other than those specified will be discarded.

- ***reqsize = [enumerated type: 1, 2, 4 and 8] default: none**

This parameter is used to specify that transactions with the specified request size are to be reported. Multiple reqsize parameters may be used to include a combination of request sizes. Transactions with reqsize values other than those specified will be discarded.

- ***ackdelay_min = [integer] default: none**
- ***ackdelay_max = [integer] default: none**

  These parameters are used to specify the range of acknowledge delays that are to be reported by the monitor. Transactions with acknowledge delays outside of this range will be discarded.

- ***reqdelay_min = [integer] default: none**
- ***reqdelay_max = [integer] default: none**

  These parameters are used to specify the range of request delays that are to be reported by the monitor. Transactions with request delays outside of this range will be discarded.

- ***unlock_min = [integer] default: none**
- ***unlock_max = [integer] default: none**

  These parameters are used to specify the range on the number of clocks (or cycles in cycle mode) that are waited before the deassertion of lock. The mode, cycle or clock, for unlock is set by the configure command unlock_mode. Transactions with unlock counts outside of the defined range will be discarded.

- ***seqaddr = [integer: 0] default: none**

  This parameter is used for the monitor to output transactions that have sequential address asserted.

- ***unseqaddr_min = [integer] default: none**
- ***unseqaddr_max = [integer] default: none**

  These parameters is used to specify the range on the number of clocks (or cycles in cycle mode) that are waited before the deassertion of seqaddr. The mode, cycle or clock, for unseqaddr is set by the configure command unseqaddr_mode. Transactions with unseqaddr counts outside of the defined range will be discarded.

- ***deselect_min = [integer] default: none**
- ***deselect_max = [integer] default: none**

  These parameters are used to specify the range on the number of clock cycles that are waited before the deassertion of select. Transactions with deselect counts outside of this range will be discarded by the monitor.

6.7.3 Read() and Write() Monitor commands

The following commands are used by the monitor to check the expected values of specified transaction attributes. If the value sampled by the monitor differs from the value that is specified by the following parameters for the current transaction, then an error will be generated.

- ***addr = [32 bit] default: none**

  This parameter is used to check the value across the address bus

- ***data = [64 bit] default: none**

This parameter is used to check the value across the data bus.

- ***ack_type = [enumerated type: normal, timeout, retry, error] default: none**

  This parameter is used to check the acknowledge type that was replied by the slave.

- ***ack_size = [enumerated type: 1, 2, 4 and 8] default: none**

  This parameter is used to check the expected full word/half word acknowledge signals of the OPB.

- ***delay = [integer] default: none**

  This parameter is used to check the number of clocks that are waited before terminating the transfer.

- ***seqaddr = [integer:0] default: none**

  This parameter is used to check if seqaddr is asserted.

- ***unseqaddr = [integer] default: none**

  This parameter is used to check the number of clocks (or cycles for cycle mode) before the deassertion of seqaddr.

- ***req_size = [enumerated type: 1, 2, 4] default: none**

  This parameter is used to check the full word/half word request signals of the OPB.

- ***req_delay = [integer] default: none**

  This parameter is used to check the number of clocks the master waited before it asserted its request signal for the corresponding read or write cycle.

- ***deselect = [integer] default: none**

  This parameter is used to check the number of clock cycles before the deassertion of select.

- ***lock = [integer: 0]**

  This parameter is used to check the number of clocks the master waited before asserting lock.

- ***unlock = [integer] default: none**

  This parameter is used to check the number of clocks (or cycles in cycle mode) the master waited before deasserting unlock.

6.7.4 Report() command

This allows the user to output the transactions that have been recorded by the monitor. Without any arguments, this command simply outputs the contents of record arrays.

- ***level = [integer: range 0 to 31] default: none**

  The level parameter indicates to output the contents of the record arrays when the level is received through the use of a send command. Multiple levels may be used in the same clock by listing more than one level. The report command outputs the transactions that have been recorded by the monitor. If no transactions have occurred across the bus, then the report statement will not output anything. It is the user's responsibility to ensure that pertinent transaction information is not lost during simulation through proper placement of report and/or send level statements.

Note: The number of transactions the monitor may record is determined by the OPB_MONITOR_RECORD_ARRAY_SIZE. Once the arrays have been filled to capacity, the

loss of transactions will occur with each transaction that it recorded. The user is responsible for determining the frequency to use the report command.

Chapter 7. OPB Bus Timing

Support for programmable setup and hold times of model signal outputs and automatic checking for valid setup and hold times of model signal inputs will be implemented in a future toolkit release. This feature would be useful in event-driven simulation environments verifying gate-level designs under test.

7.1 Examples of BFL Command Files

Three different examples are provided here to demonstrate bus functional language command files.

7.1.1 Synchronized, Unlocked Multiple Master Memory Access

This test case initializes two bus masters and one slave. Both bus masters need to access the same slave. Bus master 0 performs a sequence of write/read cycles. When all write/read cycles are completed on the bus, master 0 signals to master 1 to begin its sequence of bus cycles. Master 0 also signals to slave 2 to automatically compare its internal memory array after the write/read commands are complete.

- set_device (path=/opb_complex/opb_device0,device_type=opb_device)
  - initialize device 0
  - write(addr=00001000,be=10000000,data=12)
  - read (addr=00001000,be=10000000,data=12)
  - write(addr=00002000,be=11110000,data=00112233)
  - read (addr=00002000,be=11110000,data=00112233)
  - send (level=0)
  - send (level=1)
- set_device (path=/opb_complex/opb_device1,device_type=opb_device)
  - wait(level=1)
  - write(addr=00001000,be=1000,data=22)
  - read (addr=00001000,be=1000,data=22)
  - write(addr=00002000,be=1111,data=11112233)
  - read (addr=00002000,be=1111,data=11112233)
  - send (level=2)
- set_device (path=/opb_complex/opb_device2,device_type=opb_device)
  - configure(slave_addr_lo=00000000,slave_addr_hi=FFFFFFFF)
  - mem_init(addr=00001000,data=AABBCCDD)
  - mem_init(addr=00002000,data=AABBCCDD)
  - response(delay=2,ack_type=normal,ack_size=1)

- response(delay=1,ack_type=normal,ack_size=1)
- response(delay=3,ack_type=normal,ack_size=4)
- response(delay=4,ack_type=normal,ack_size=4)
- response(delay=2,ack_type=normal,ack_size=1)
- response(delay=1,ack_type=normal,ack_size=1)
- response(delay=3,ack_type=normal,ack_size=4)
- response(delay=4,ack_type=normal,ack_size=4)
- mem_check(level=0,addr=00001000,data=12BBCCDD)
- mem_check(level=0,addr=00002000,data=00112233)
- mem_check(level=2,addr=00001000,data=22BBCCDD)
- mem_check(level=2,addr=00002000,data=11112233)

7.1.2 Locked Multiple Master Memory Access

This test case initializes two bus masters and one slave. Both bus masters need to access the same slave. Parking is disabled (default). Arbiter mode is priority. Bus masters 0 and 1 request access to slave 2. Master 0 requests the bus. Arbiter grants bus to master 0 since it's the only request. Master 0 asserts busLock with select in clock 1; Master 1 requests the bus one clock after master 1; Although master 1 has higher priority in the arbiter, it doesn't get granted.

– set_device (path=/opb_complex/opb_arbiter,device_type=opb_device)
  - configure(arbiter_mode=priority)
– set_device (path=/opb_complex/opb_device0,device_type=opb_device)
  - initialize master 0
  - write(addr=00021000,req_size=1,data=12,lock=0,unlock_mode=cycle,unlock=4)
  - read (addr=00021000,req_size=1,data=12)
  - write(addr=00022000,req_size=4,data=00112233)
  - read (addr=00022000,req_size=4,data=00112233)
  - send (level=0)
– set_device (path=/opb_complex/opb_device1,device_type=opb_device)
  - initialize master 1
  - write(delay=1,addr=00021000,req_size=1,data=22)
  - read (addr=00021000,req_size=1,data=22)
  - write(addr=00022000,req_size=4,data=11112233)
  - read (addr=00022000,req_size=4,data=11112233)
  - send(level=2)
– set_device (path=/opb_complex/opb_device2,device_type=opb_device)
  - initialize slave 2

- mem_init(addr=00021000,data=AABBCCDD)
- mem_init(addr=00022000,data=AABBCCDD)
- response(delay=2,ack_type=normal,ack_size=4)
- response(delay=1,ack_type=normal,ack_size=4)
- response(delay=3,ack_type=normal,ack_size=4)
- response(delay=4,ack_type=normal,ack_size=4)
- response(delay=0,ack_type=normal,ack_size=4)
- response(delay=1,ack_type=normal,ack_size=4)
- response(delay=3,ack_type=normal,ack_size=4)
- response(delay=2,ack_type=normal,ack_size=4)
- mem_check(level=0,addr=00021000,data=12BBCCDD)
- mem_check(level=0,addr=00022000,data=00112233)
- mem_check(level=2,addr=00021000,data=2222CCDD)
- mem_check(level=2,addr=00022000,data=11112233)

7.1.3 Burst Write/Read with Lock and SeqAddr

This test case initializes a bus master and a slave. Parking is disabled. Arbiter mode is round robin. The bus master burst writes 16 bytes, and then reads it back. The bus remains locked for eight cycles, and seqaddr is asserted. The default cycle count unlock/unseqaddr mode is used

– set_device (path=/opb_complex/opb_arbiter,device_type=opb_device)
  - initialize arbiter
  - configure(arbiter_mode=round)
– set_device (path=/opb_complex/opb_device0,device_type=opb_device)
  - initialize master 0
  - write (addr=00021000,req_size=4,data=00112233,seqaddr=0,unseqaddr=4,lock=0,unlock=8)
  - write (addr=00021004,req_size=4,data=44556677)
  - write (addr=00021008,req_size=4,data=8899AABB)
  - write (addr=0002100C,req_size=4,data=CCDDEEFF)
  - read (addr=00021000,req_size=4,data=00112233,seqaddr=0,unseqaddr=4)
  - read (addr=00021004,req_size=4,data=44556677)
  - read (addr=00021008,req_size=4,data=8899AABB)
  - read (addr=0002100C,req_size=4,data=CCDDEEFF)
  - send(level=0)
– set_device (path=/opb_complex/opb_device2,device_type=opb_device)
  - initialize slave 2
  - mem_init(addr=00021000,data=AABBCCDD)

- mem_init(addr=00021004,data=AABBCCDD)
- mem_init(addr=00021008,data=AABBCCDD)
- mem_init(addr=0002100C,data=AABBCCDD)
- response(delay=4,ack_type=normal,ack_size=4)
- response(delay=1,ack_type=normal,ack_size=4)
- response(delay=1,ack_type=normal,ack_size=4)
- response(delay=1,ack_type=normal,ack_size=4)
- response(delay=4,ack_type=normal,ack_size=4)
- response(delay=1,ack_type=normal,ack_size=4)
- response(delay=1,ack_type=normal,ack_size=4)
- response(delay=1,ack_type=normal,ack_size=4)
- mem_check(level=0,addr=00021000,data=00112233)
- mem_check(level=0,addr=00021004,data=44556677)
- mem_check(level=0,addr=00021004,data=8899AABB)
- mem_check(level=0,addr=00021004,data=CCDDEEFF)

7.1.4 Burst Write/Read with Lock and SeqAdd with Byte Enables

This test case initializes a bus master and a slave. Parking is disabled. Arbiter mode is round robin. The bus master burst writes 16 bytes, and then reads it back. The bus remains locked for eight cycles, and seqaddr is asserted. The default cycle count unlock/unseqaddr mode is used

– set_device (path=/opb_complex/opb_arbiter,device_type=opb_device)
  - initialize arbiter
  - configure(arbiter_mode=round)
– set_device (path=/opb_complex/opb_device0,device_type=opb_device)
  - initialize master 0
  - write (addr=00021000,req_size=4,be=11110000, data=00112233,seqaddr=0,unseqaddr=4,lock=0,unlock=8)
  - write (addr=00021004,req_size=4,be=11110000,data=44556677)
  - write (addr=00021008,req_size=4,be=11110000,data=8899AABB)
  - write (addr=0002100C,req_size=4,be=11110000,data=CCDDEEFF)
  - read (addr=00021000,req_size=4,be=11110000,data=00112233,seqaddr=0,unseqaddr=4)
  - read (addr=00021004,req_size=4,be=11110000,data=44556677)
  - read (addr=00021008,req_size=4,be=11110000,data=8899AABB)
  - read (addr=0002100C,req_size=4,be=11110000,data=CCDDEEFF)
– send(level=0)
– set_device (path=/opb_complex/opb_device2,device_type=opb_device)

– configure(byte_enable=1)
  - initialize slave 2
  - mem_init(addr=00021000,data=AABBCCDD)
  - mem_init(addr=00021004,data=AABBCCDD)
  - mem_init(addr=00021008,data=AABBCCDD)
  - mem_init(addr=0002100C,data=AABBCCDD)
  - response(delay=4,ack_type=normal,ack_size=4)
  - response(delay=1,ack_type=normal,ack_size=4)
  - response(delay=1,ack_type=normal,ack_size=4)
  - response(delay=1,ack_type=normal,ack_size=4)
  - response(delay=4,ack_type=normal,ack_size=4)
  - response(delay=1,ack_type=normal,ack_size=4)
  - response(delay=1,ack_type=normal,ack_size=4)
  - response(delay=1,ack_type=normal,ack_size=4)
  - mem_check(level=0,addr=00021000,data=00112233)
  - mem_check(level=0,addr=00021004,data=44556677)
  - mem_check(level=0,addr=00021004,data=8899AABB)
  - mem_check(level=0,addr=00021004,data=CCDDEEFF)

7.1.5 Write/Read with Transaction Checking and Reporting

This test case initializes a bus master, slave, and monitor. Arbiter mode is round robin. The bus master performs four write followed by four read cycles. The bus monitor is configured to cycle mode for unlock checking, and to report recorded transactions when a level of five is received. In addition, this test checks the integrity of every transaction that is simulated with the read and write statements in the monitor section. Every parameter in these commands is optional. If any of the specified parameters does not match the activity the bus records, then errors will be output during simulation.

– set_device (path=/opb_complex/opb_arbiter, device_type=opb_arbiter)
  - configure(arbiter_mode=round)
– set_device (path=/opb_complex/opb_device0, device_type=opb_device)
  - configure(unlock_mode=cycle)
  - write (addr=00021000, req_size=4, req_delay=4, lock=0, unlock=1, data=00112233)
  - write (addr=00021004, req_size=4, req_delay=4, data=44556677)
  - write (addr=00021008, req_size=4, req_delay=4, data=8899AABB)
  - write (addr=0002100C, req_size=4, req_delay=4, data=CCDDEEFF)
  - read (addr=00021000, req_size=4, req_delay=4, lock=0, unlock=2, data=00112233)
  - read (addr=00021004, req_size=4, req_delay=4, data=44556677)
  - read (addr=00021008, req_size=4, req_delay=4, data=8899AABB)

- read (addr=0002100C, req_size=4, req_delay=4, data=CCDDEEFF)
- send (level=5)

– set_device (path=/opb_complex/opb_device2, device_type=opb_device)
- configure(slave_addr_hi_0=10000000,slave_addr_lo_0=00000000,
- slave_addr_hi_1=30000000,slave_addr_lo_1=20000000)
- mem_init (addr=00021000, data=AABBCCDD)
- mem_init (addr=00021004, data=AABBCCDD)
- mem_init (addr=00021008, data=AABBCCDD)
- mem_init (addr=0002100C, data=AABBCCDD)
- response (delay=4, ack_type=normal, ack_size=4)
- response (delay=1, ack_type=normal, ack_size=4)
- response (delay=1, ack_type=normal, ack_size=4)
- response (delay=1, ack_type=normal, ack_size=4)
- response (delay=4, ack_type=normal, ack_size=4)
- response (delay=1, ack_type=normal, ack_size=4)
- response (delay=1, ack_type=normal, ack_size=4)
- response (delay=1, ack_type=normal, ack_size=4)

– set_device (path=/opb_complex/opb_mon, device_type=opb_monitor)
- configure (unlock_mode=cycle)
- report (level=5)
- write (addr=00021000, req_size=4, req_delay=4, lock=0, unlock=1, data=00112233, delay=4, ack_type=normal)
- write (addr=00021004, req_size=4, req_delay=4, data=44556677, delay=1, ack_type=normal)
- write (addr=00021008, req_size=4, req_delay=4, data=8899AABB, delay=1, ack_type=normal)
- write (addr=0002100C, req_size=4, req_delay=4, data=CCDDEEFF, delay=1, ack_type=normal)
- read (addr=00021000, req_size=4, req_delay=4, lock=0, unlock=2, data=00112233, delay=4, ack_type=normal)
- read (addr=00021004, req_size=4, req_delay=4, data=44556677, delay=1, ack_type=normal)
- read (addr=00021008, req_size=4, req_delay=4, data=8899AABB, delay=1, ack_type=normal)
- read (addr=0002100C, req_size=4, req_delay=4, data=CCDDEEFF, delay=1, ack_type=normal)

Chapter 8. OPB Bus Compliance Checks

8.1 Terminology

The following terminology is used in the specification of the OPB monitor compliance checks:

- Active: refers to the situation when a signal is at its true state (be it high or low).
- Asserted: refers to the situation when a signal transitions from inactive to its active state.
- Inactive: refers to the situation when a signal is at its false state.
- Deasserted: refers to the situation when a signal transitions from active to its inactive state.

8.2 OPB Signal Summary Table

Table 1 provides a summary of all OPB input/output signals in alphabetical order which includes the source, a brief description and page reference for bus compliance checks. See OPB architecture specifications for detailed signal description.

Table 1. Summary of OPB Signals

Signal Name	Source	Description	Page
DMA_SlnAck	DMA	DMA slave acknowledge	56
Mn_BE(0:7)	Master	Master address bus	49
Mn_BEXFER	Master	Master address bus	49
Mn_UABus(0:31)	Master	Master address bus	49
Mn_ABus(0:31)	Master	Master address bus	53
Mn_busLock	M/A	Master bus arbitration lock	47
Mn_DBus(0:63)	Master	Master data bus	49
Mn_DBusEn	Master	Master data bus enable	49
Mn_dwXfer	Master	Master doubleword transfer	48
Mn_fwXfer	Master	Master fullword transfer	48
Mn_hwXfer	Master	Master halfword transfer	48
Mn_RNW	Master	Master read not write	48
Mn_select	Master	Master select	47
Mn_seqAddr	Master	Master sequential address	48
OPB_UABus(0:31)	M/S	OPB address bus upper	50
OPB_ABus(0:31)	M/S	OPB address bus	50
OPB_busLock	Arbiter	OPB bus arbitration lock	50

Table 1. Summary of OPB Signals (Continued)

Signal Name	Source	Description	Page
OPB_DBus(0:64)	M/S	OPB data bus	53
OPB_errAck	Master	OPB error acknowledge	51
OPB_dwAck	Slave	OPB doubleword acknowledge	51
OPB_dwXfer	Master	OPB doublelword transfer	52
OPB_fwAck	Slaver	OPB fullword acknowledge	51
OPB_fwXfer	Master	OPB fullword transfer	52
OPB_hwAck	Slave	OPB halfword acknowledge	53
OPB_hwXfer	Master	OPB halfword transfer	52
OPB_BEXfert	M	OPB master byte enable transfer	50
OPB_BE	M	OPB master byte enable vector	50
OPB_MnGrant	M/A	OPB master bus grant	47
OPB_retry	Master	OPB bus cycle retry	51
OPB_RNW	Master	OPB read not write	52
OPB_select	M/A	OPB select	52
OPB_seqAddr	Master	OPB sequential address	53
OPB_timeout	M/A	OPB timeout error	47
OPB_toutSup	Arbiter	OPB timeout suppress	55
OPB_xferAck	M/A	OPB transfer acknowledge	50
Sln_DBus(0:63)	Slave	Slave data bus	55
Sln_DBusEn	Slave	Slave data bus enable	55
Sln_DMAReq	DMA	Slave DMA request	56
Sln_errAck	Slave	Slave error acknowledge	54
Sln_dwAck	Slave	Slave fullword acknowledge	54
Sln_fwAck	Slave	Slave fullword acknowledge	54
Sln_hwAck	Slave	Slave halfword acknowledge	53
Sln_retry	Slave	Slave bus cycle retry	54
Sln_toutSup	Slave	Slave timeout suppress	55
Sln_xferAck	Slave	Slave transfer acknowledge	53

8.3 OPB Master Interface Checks

8.3.1 OPB_MnGrant

The following error messages are issued for this signal:

- Error 1.2.1

  Check for only one OPB_MnGrant active in any cycle.

- Error 1.2.2

  Check for OPB_MnGrant to be active only for owning master.

8.3.2 OPB_timeout

The following error messages are issued for this signal:

- Error 1.5.1

  Check that OPB_timeout is asserted only after 16 cycles have elapsed without any response from a slave device (i.e. no OPB_xferAck or OPB_retry), since the assertion of OPB_select. The count may be suppressed by OPB_toutSup from the addressed slave. NOTE: The 16 cycles are not necessarily in succession.

- Error 1.5.2

  Check that while OPB_select is inactive there must be no responses from slave devices. OPB_timeout must be inactive.

8.3.3 Mn_busLock

The following error messages are issued for this signal:

- Error 1.3.1

  Check that when OPB_busLock is asserted that OPB_select is active or asserted.

- Error 1.3.2

  Check that when OPB_busLock is active the only active OPB_MnGrant should correspond to the locking master.

8.3.4 Mn_select

The following error messages are issued for this signal:

- Error 1.8.1

  Check that when OPB_select is asserted, OPB_MnGrant or OPB_busLock were active in the previous cycle.

- Error 1.8.2

  Check that only one Mn_select is active in any cycle.

8.3.5 Mn_RNW

The following error messages are issued for this signal:

- Error 1.9.1

Check that when OPB_select is inactive that OPB_RNW is inactive.

8.3.6 Mn_hwXfer, Mn_fwXfer, Mn_dwXfer

The following error messages are issued for this signal:

- Error 1.10.1

  Check that while OPB_select is inactive that OPB_hwXfer, OPB_fwXfer, and OPB_dwXfer are inactive.

- Error 1.10.2

  Transfer size encode of OPB_hwXfer, OPB_fwXfer, and OPB_dwXfer(3'b001, 3'b010, 3'b011, or 3'b101) is reserved and will be checked when OPN_select is active.

- Error 1.10.3

  Check address and requested transfer size against invalid combinations of OPB_ABus(29,30,31) and transfer size as indicated in Table 2.

Table 2. Unaligned Transfers Checked

OPB_ABus(29,30, 31)	OPB_hwXfer	OPB_fwXfer	OPB_dwXfer	Transfer Size
(X,X,1)	1	0	0	Halfword
(X,X,1)	1	1	0	Fullword
(X,1,0)	1	1	0	Fullword
(X,X,1)	1	1	1	Doubleword
(X,1,0)	1	1	1	Doubleword
(1,0,0)	1	1	1	Doubleword

8.3.7 Mn_seqAddr

The following error messages are issued for this signal:

- Error 1.11.1

  Check that OPB_seqAddr is active or asserted only when BusLock is active.

- Error 1.11.2

  Check that while OPB_select is inactive that OPB_seqAddr must be inactive.

- Error 1.11.3

Check that when OPB_seqAddr is active the OPB_ABus incremented sequentially from the previous transfer. The increment amount will be the smallest size decoded from OPB_dwXfer, OPB_fwXfer, OPB_hwXfer vs. OPB_dwAck, OPB_fwAck, OPB_hwAck signal values of the current operation.

- Error 1.11.4

  Check that OPB_seqAddr is only asserted with the assertion of select or after a xferack.

8.3.8 Mn_UAbus (0:31) and Mn_ABus(0:31)

The following error messages are issued for this signal:

- Error 1.6.1

  Check that while OPB_select is inactive, there must be zeroes on OPB_ABus.

8.3.9 Mn_BE and Mn_BEXFER

- Error 1.20.1

  Byte enable transfer initiated by asserting BEXFER without any byte lanes asserted.

- Error 1.20.2

  Byte lanes active without BEXFER signal active.

- Error 1.20.3

  Address offset is not aligned with the first active byte lane of the byte-enable bus.

- Error 1.20.4

  Non contiguous byte lanes have been detected.

8.3.10 Mn_DBus(0:63)

The following error messages are issued for this signal:

- Error 1.7.1

  Check that while OPB_select is inactive, there must be zeroes on OPB_ABus.

8.3.11 Mn_DBusEn

The following error messages are issued for this signal:

- Error1.12.1

  Check during a write transfer that Mn_DBusEn is high starting with the assertion of Mn_select for 32 bit transfers and MnDBusEn32_63 is also asserted when transfers are greater than 32 bits.

- Error 1.12.2

  Check during a read transfer that Sln_DBusEn is high starting with the assertion of Mn_select for 32 bit transfers and SLnDBusEn32_63 is also asserted when transfers are greater than 32 bits.

8.3.12 OPB_UAbus and OPB_ABus(0:31)

The following error messages are issued for this signal:

- Error 1.6.1

  Check that while OPB_select is inactive, there must be zeroes on OPB_ABus.

8.3.13 OPB_BE and OPB_BEXFER

- Error 1.20.1

  Byte enable transfer initiated by asserting BEXFER without any byte lanes asserted.

- Error 1.20.2

  Byte lanes active without BEXFER signal active.

- Error 1.20.3

  Address offset is not aligned with the first active byte lane of the byte-enable bus.

- Error 1.20.4

  Non contiguous byte lanes have been detected.

8.3.14 OPB_DBus(0:63)

The following error messages are issued for this signal:

- Error 1.7.1

  Check that while OPB_select is inactive, there must be zeroes on OPB_DBus.

8.3.15 OPB_xferAck

The following error messages are issued for this signal:

- Error 1.13.1

  Check that OPB_xferAck is asserted only when OPB_select is active.

- Error 1.13.2

  Check that OPB_xferAck is asserted before 16 cycles have elapsed since the assertion of OPB_select. Note that the 16 cycles are not necessarily successive. The count may be suppressed by OPB_ToutSup.

8.3.16 OPB_hwAck

The following error messages are issued for this signal:

- Error 1.14.1

  Check that while OPB_select is inactive that OPB_hwAck must be inactive.

- Error 1.14.2

  Transfer Acknowledge encode of OPB_hwAck high is reserved and will be checked when OPB_xferAck is active.

8.3.17 OPB_fwAck

The following error messages are issued for this signal:

- Error 1.14.1

  Check that while OPB_select is inactive that OPB_fwAck must be inactive.

- Error 1.14.2

  Transfer Acknowledge encode of OPB_fwAck high is reserved and will be checked when OPB_xferAck is active.

8.3.18 OPB_dwAck

The following error messages are issued for this signal:

- Error 1.14.1

  Check that while OPB_select is inactive that OPB_dwAck must be inactive.

- Error 1.14.2

  Transfer Acknowledge encode of OPB_dwAck high is reserved and will be checked when OPB_xferAck is active

8.3.19 OPB_errAck

The following error messages are issued for this signal:

- Error 1.15.1

  Check that while OPB_select is inactive that OPB_errAck must be inactive.

8.3.20 OPB_retry

The following error messages are issued for this signal:

- Error 1.4.0

  Check that OPB_retry and OPB_xferAck are not active at the same time.

- Error 1.4.1

  Check that upon a valid OPB_retry the response is to deassert OPB_busLock (if active), keep it inactive for one cycle.

- Error 1.4.2

  Check that upon a valid OPB_retry that the response is to deassert Mn_request (if active) and keep it inactive for one cycle.

- Error 1.4.3

  Check that upon a valid OPB_retry that the OPB_select is deasserted (which must be the next cycle, i.e OPB_retry is active for only 1 cycle).

- Error 1.4.4

  Check that while OPB_select is inactive, that OPB_retry must be inactive.

8.4 OPB Slave Interface Checks

8.4.1 OPB_select

The following error messages are issued for this signal:

- Error 1.8.1

  Check that when OPB_select is asserted, OPB_MnGrant or OPB_busLock were active in the previous cycle.

- Error 1.8.2

  Check that only one Mn_select is active in any cycle.

8.4.2 OPB_RNW

The following error messages are issued for this signal:

- Error 1.9.1

  Check that while OPB_select is active that OPB_RNW must be inactive.

8.4.3 OPB_hwXfer, OPB_fwXfer and OPB_dwXfer

The following error messages are issued for this signal:

- Error 1.10.1

  Check that while OPB_select is inactive that OPB_hwXfer, OPB_fwXfer, and OPB_dwXfer are inactive.

- Error 1.10.2

  Transfer size encode of OPB_hwXfer, OPB_fwXfer, and OPB_dwXfer(3’b001, 3’b010, 3’b011, or 3’b101) is reserved and will be checked when OPN_select is active.

  Error 1.10.3Check address and requested transfer size against invalid combinations of OPB_ABus(29,30,31) and transfer size as indicated in Table 3.

Table 3. Unaligned Transfers Checked

OPB_ABus(29,30, 31)	OPB_hwXfer	OPB_fwXfer	OPB_dwXfer	Transfer Size
(X,X,1)	1	0	0	Halfword
(X,X,1)	1	1	0	Fullword
(X,1,0)	1	1	0	Fullword
(X,X,1)	1	1	1	Doubleword
(X,1,0)	1	1	1	Doubleword
(1,0,0)	1	1	1	Doubleword

8.4.4 OPB_seqAddr

The following error messages are issued for this signal:

- Error 1.11.1

  Check that OPB_seqAddr is active or asserted only when BusLock is active.

- Error 1.11.2

  Check that while OPB_select is inactive that OPB_seqAddr must be inactive.

- Error 1.11.3

  Check that when OPB_seqAddr is active the OPB_ABus incremented sequentially from the previous transfer. The increment amount will be the smallest size decoded from OPB_dwXfer, OPB_fwXfer, OPB_hwXfer vs. OPB_dwAck, OPB_fwAck, OPB_hwAck signal values of the current operation.

8.4.5 OPB_UABus(0:31) and OPB_ABus(0:31)

The following error messages are issued for this signal:

- Error 1.6.1

  Check that while OPB_select is inactive, there must be zeroes on OPB_ABus.

8.4.6 OPB_DBus(0:63)

The following error messages are issued for this signal:

- Error 1.7.1

  Check that while OPB_select is inactive, there must be zeroes on OPB_DBus.

8.4.7 Sln_xferAck

The following error messages are issued for this signal:

- Error 1.13.1

  Check that OPB_xferAck is asserted only when OPB_select is active.

- Error 1.13.2

  Check that OPB_xferAck is asserted before 16 cycles have elapsed since the assertion of OPB_select. Note that the 16 cycles are not necessarily successive. The count may be suppressed by OPB_ToutSup.

8.4.8 Sln_hwAck

The following error messages are issued for this signal:

- Error 1.14.1

  Check that while OPB_select is inactive that OPB_hwAck must be inactive.

- Error 1.14.2

Transfer Acknowledge encode of OPB_hwAck high is reserved and will be checked when OPB_xferAck is active.

8.4.9 Sln_fwAck

The following error messages are issued for this signal:

- Error 1.14.1

  Check that while OPB_select is inactive that OPB_fwAck must be inactive.

- Error 1.14.2

  Transfer Acknowledge encode of OPB_fwAck high is reserved and will be checked when OPB_xferAck is active.

8.4.10 Sln_dwAck

The following error messages are issued for this signal:

- Error 1.14.1

  Check that while OPB_select is inactive that OPB_dwAck must be inactive.

- Error 1.14.2

  Transfer Acknowledge encode of OPB_dwAck high is reserved and will be checked when OPB_xferAck is active.

8.4.11 Sln_errAck

The following error messages are issued for this signal:

- Error 1.15.1

  Check that while OPB_select is inactive there must be no response from slave devices and OPB_errAck must be inactive.

8.4.12 Sln_retry

The following error messages are issued for this signal:

- Error 1.4.0

  Check that OPB_retry and OPB_xferAck are not active at the same time.

- Error 1.4.1

  Check that upon a valid OPB_retry the response is to deassert OPB_busLock (if active), keep it inactive for one cycle.

- Error 1.4.2

  Check that upon a valid OPB_retry that the response is to deassert Mn_request (if active) and keep it inactive for one cycle.

- Error 1.4.3

Check that upon a valid OPB_retry that the OPB_select is deasserted (which must be the next cycle, i.e OPB_retry is active for only 1 cycle).

- Error 1.4.4

  Check that while OPB_select is inactive, that OPB_retry must be inactive.

8.4.13 Sln_toutSup

The following error messages are issued for this signal:

- 1.16.1

  Check that while OPB_select is inactive that OPB_toutSup must be inactive.

8.4.14 Sln_DBus(0:63)

The following error messages are issued for this signal:

- Error 1.7.1

  Check that while OPB_select is inactive, there must be zeroes on OPB_DBus.

8.4.15 Sln_DBusEn

The following error messages are issued for this signal:

- Error1.12.1

  Check during a write transfer that Mn_DBusEn and/or MnDBusEn32_63 is high starting with the assertion of Mn_select.

- Error 1.12.2

  Check during a read transfer that Sln_DBusEn and/or MnDBusEn32_63 is high starting with the assertion of Mn_select.

8.5 OPB Arbiter Interface Checks

8.5.1 OPB_busLock

The following error messages are issued for this signal:

- Error 1.3.1

  Check that when OPB_busLock is asserted that OPB_select is active or asserted.

- Error 1.3.2

  Check that when OPB_busLock is active the only active OPB_MnGrant should correspond to the locking master.

8.5.2 OPB_toutSup

The following error messages are issued for this signal:

- Error 1.16.1

Check that while OPB_select is inactive there must be no response from slave devices. OPB_errAck must be inactive.

8.6 OPB DMA Interface Checks

8.6.1 DMA_SlnAck

The following error messages are issued for this signal:

- Error 1.17.1

  Check that DMA_Sl[y]Ack is never asserted without Sl[y]_DMAReq active.

8.7 Dynamic Bus Sizing (DBS)

Data mirroring on appropriate byte lanes during DBS should occur and are checked according to the following tables

- Error 1.18.1

Checks for proper data mirroring during byte enabled writes.

Table 4. 64-bit Master Write Data Mirroring During Byte Enable Writes

			64-bit Data Bus							
ABus 29:31	**Mn_BE (0:7)**	**Request Transfer Size**	**Dbus 0:7 byte0**	**Dbus 8:15 byte1**	**Dbus 16:23 byte2**	**Dbus 24:31 byte3**	**Dbus 32:39 byte4**	**Dbus 40:47 byte5**	**Dbus 48:55 byte6**	**Dbus 56:63 byte7**
000	1111_1111	doubleword	byte0	byte1	byte2	byte3	byte4	byte5	byte6	byte7
000	1111_1110	fullword	byte0	byte1	byte2	byte3	byte4	byte5	byte6	
001	0111_1111	byte	**byte1**	byte1	byte2	byte3	byte4	byte5	byte6	byte7
000	1111_1100	fullword	byte0	byte1	byte2	byte3	byte4	byte5		
001	0111_1110	byte	**byte1**	byte1	byte2	byte3	byte4	byte5	byte6	
010	0011_1111	halfword	**byte2**	**byte3**	byte2	byte3	byte4	byte5	byte6	byte7
000	1111_1000	fullword	byte0	byte1	byte2	byte3	byte4			
001	0111_1100	byte	**byte1**	byte1	byte2	byte3	byte4	byte5		
010	0011_1110	halfword	**byte2**	**byte3**	byte2	byte3	byte4	byte5	byte6	
011	0001_1111	byte	**byte3**	**byte3**		byte3	byte4	byte5	byte6	byte7
000	1111_0000	fullword	byte0	byte1	byte2	byte3				
001	0111_1000	byte	**byte1**	byte1	byte2	byte3	byte4			
010	0011_1100	halfword	**byte2**	**byte3**	byte2	byte3	byte4	byte5		
011	0001_1110	byte	**byte3**	**byte3**		byte3	byte4	byte5	byte6	

Table 4. 64-bit Master Write Data Mirroring During Byte Enable Writes (Continued)

			64-bit Data Bus							
ABus 29:31	**Mn_BE (0:7)**	**Request Transfer Size**	**Dbus 0:7 byte0**	**Dbus 8:15 byte1**	**Dbus 16:23 byte2**	**Dbus 24:31 byte3**	**Dbus 32:39 byte4**	**Dbus 40:47 byte5**	**Dbus 48:55 byte6**	**Dbus 56:63 byte7**
100	0000_1111	fullword	**byte4**	**byte5**	**byte6**	**byte7**	byte4	byte5	byte6	byte7
000	1110_0000	halfword	byte0	byte1	byte2					
001	0111_0000	byte	**byte1**	byte1	byte2	byte3				
010	0011_1000	halfword	**byte2**	**byte3**	byte2	byte3	byte4			
011	0001_1100	byte	**byte3**	**byte3**		byte3	byte4	byte5		
100	0000_1110	halfword	**byte4**	**byte5**	**byte6**		byte4	byte5	byte6	
101	0000_0111	byte	**byte5**	**byte5**	**byte6**	**byte7**		byte5	byte6	byte7
000	1100_0000	halfword	byte0	byte1						
001	0110_0000	byte	**byte1**	byte1	byte2					
010	0011_0000	halfword	**byte2**	**byte3**	byte2	byte3				
011	0001_1000	byte	**byte3**	**byte3**		byte3	byte4			
100	0000_1100	doublewor d	**byte4**	**byte5**			byte4	byte5		
101	0000_0110	byte	**byte5**	**byte5**	**byte6**			byte5	byte6	
110	0000_0011	halfword	**byte6**	**byte7**	**byte6**	**byte7**			byte6	byte7
000	1000_0000	byte	byte0							
001	0100_0000	byte	**byte1**	byte1						
010	0010_0000	byte	**byte2**		byte2					
011	0001_0000	byte	**byte3**	**byte3**		byte3				
100	0000_1000	byte	**byte4**				byte4			
101	0000_0100	byte	**byte5**	**byte5**				byte5		
110	0000_0010	byte	**byte6**		**byte6**				byte6	
111	0000_0001	byte	**byte7**	**byte7**		**byte7**				byte7

- Error 1.18.1

  Checks for proper data mirroring during non byte enabled writes.

Table 5. 64-bit Master Write Data Mirroring During Non Byte Enable Writes

		64-bit Data Bus							
ABus (29:31)	**Transfer Size**	**Dbus 0:7 byte0**	**Dbus 8:15 byte1**	**Dbus 16:23 byte2**	**Dbus 24:31 byte3**	**Dbus 32:39 byte4**	**Dbus 40:47 byte5**	**Dbus 48:55 byte6**	**Dbus 56:63 byte7**
000	doubleword	byte0	byte1	byte2	byte3	byte4	byte5	byte6	byte7
000	fullword	byte0	byte1	byte2	byte3				
100	fullword	**byte4**	**byte5**	**byte6**	**byte7**	byte4	byte5	byte6	byte7
000	halfword	byte0	byte1						
010	halfword	**byte2**	**byte3**	byte2	byte3				
100	halfword	**byte4**	**byte5**			byte4	byte5		
110	halfword	**byte6**	**byte7**	**byte6**	**byte7**			byte6	byte7
000	byte	byte0							
001	byte	**byte1**	byte1						
010	byte	**byte2**		byte2					
011	byte	**byte3**	**byte3**		byte3				
100	byte	**byte4**				byte4			
101	byte	**byte5**	**byte5**				byte5		
110	byte	**byte6**		**byte6**				byte6	
111	byte	**byte7**	**byte7**		**byte7**				byte7

8.8 64-bit Read Data Steering

Read data steering must be performed by 64 bit slaves when address bit 29 is a one to support access by 32 masters. The following table illustrates proper steering of bytes.

- Error 1.18.2

  A 64 bit acknowledge was detected and proper read data steering was violated.

Table 6. 64-bit Slave Read Steering

		64-bit Data Bus							
ABus (29:31)	**Transfer Size**	**Dbus 0:7 byte0**	**Dbus 8:15 byte1**	**Dbus 16:23 byte2**	**Dbus 24:31 byte3**	**Dbus 32:39 byte4**	**Dbus 40:47 byte5**	**Dbus 48:55 byte6**	**Dbus 56:63 byte7**
000	doubleword	byte0	byte1	byte2	byte3	byte4	byte5	byte6	byte7
000	fullword	byte0	byte1	byte2	byte3				
100	fullword	**byte4**	**byte5**	**byte6**	**byte7**	byte4	byte5	byte6	byte7
000	halfword	byte0	byte1						
010	halfword			byte2	byte3				
100	halfword	**byte4**	**byte5**			byte4	byte5		
110	halfword			**byte6**	**byte7**			byte6	byte7
000	byte	byte0							
001	byte		byte1						
010	byte			byte2					
011	byte				byte3				
100	byte	**byte4**				byte4			
101	byte		**byte5**				byte5		
110	byte			**byte6**				byte6	
111	byte				**byte7**				byte7

8.9 Address/Control/Data Hold Checks

The following error messages are issued for this signal:

- Error 1.19.1

  Check that OPB_ABus does not change while OPB_select is active and OPB_xferAck is inactive.

- Error 1.19.2

  Check that OPB_DBus does not change while OPB_select is active and OPB_xferAck is inactive and OPB_RNW is low indicating a write operation.

- Error 1.19.3

  Check that OPB_hwXfer, OPB_fwXfer and OPB_RNW do not change while OPB_select is active and OPB_xferAck is inactive.

- Error 1.19.4

  Check that OPB_select does not become inactive before receipt of OPB_xferAck.

- Error 1.19.5

  Check that OPB_BE does not change state during a read or write before receipt of a transfer acknowledge.

Index

Numerics

64-bit read data steering 59

A

about this book xiii
address checks 60
alias commands 23
ALU intructions 17

B

branch command 17
bus compliance checks 45
bus functional compiler 10
bus functional compiler and informational files 6
bus functional language 21
  OPB toolkit
    configuration commands 21
bus models 12

C

compiler simulator configuration 10
control checks 60

D

data checks 60
dynamic bus sizing 56

E

examples of bfl command files 39

G

general purpose registers 17

H

hold checks 60

I

ieee packages 9
initializing bus functional models 10
instantiating design under test 8
invoking bus functional compiler 10

M

master modes 21
model operation 16

O

OPB arbiter checks 55
OPB arbiter master interface 18
OPB arbiter models 17
OPB arbiter operation 18
OPB bus features 3
OPB data interface 16
OPB device configuration commands 22
OPB device model 13
OPB device models 21
OPB DMA checks 56
OPB DMA interface 16
OPB implementation 4
OPB master checks 47
OPB master commands 24
OPB master interface 14
OPB model testbench 8
OPB monitor 19
OPB monitor commands 34
OPB monitor interface 19
OPB signal summary 45
OPB slave checks 52
OPB slave commands 31
OPB slave interface 15
OPB timing 39
OPB toolkit 1
OPB toolkit environment 6
OPB toolkit features 2

S

script files 6
send commands 23
slave modes 22
synchronization commands 23

V

verilog files 7
vhdl files 7
vhdl signal types 9

W

wait commands 23

© International Business Machines Corporation 1996, 2000
Printed in the United States of America
5/2/01
All Rights Reserved

The information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change IBM's product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as illustration. The results obtained in other operating environments may vary.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS IS" BASIS. In no event will IBM be liable for any damages arising directly or indirectly from any use of the information contained in this document.

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY
12533-6531

The IBM home page can be found at http://www.ibm.com

The IBM Microelectronics Division home page can be found at
http://www.chips.ibm.com

Document No. SA-14-2541-02